URL: http://www.elsevier.nl/locate/entcs/volume23.html 19 pages

Model-Generation of a Fictitious Clock Real-Time Logic Using Sharing Trees

Laurent Ferier,[a] Jean-François Raskin,[b,c]
Pierre-Yves Schobbens[a]

[a] *Institut d'Informatique - F.U.N.D.P. - Namur - Belgium*

[b] *Max-Planck Institut für Informatik - Saarbrücken - Germany*

[c] *EECS - University of California at Berkeley - USA*

Abstract

We present first the logic MTL, a real-time temporal logic that is at the heart of the real-time specification language Albert. Since this logic is undecidable, we approximate it (using the theory of Abstract Interpretation) by its fictitious clock counterpart, $\mathsf{MTL}^{\mathsf{FC}}$.

We then present a symbolic tableau-based *model generation* decision procedure in ExpSpace, which is theoretically optimal. In practice however, we see that the introduction of integer-valued *prophecy variables* will make it more efficient. From these variables, we reconstruct by reverting the process a logic that we call $\mathsf{ECL}^{\mathsf{FC}}$, which can be decided in PSpace, and has the same expressivity as $\mathsf{MTL}^{\mathsf{FC}}$.

Theory thus shows that memory space is the critical factor. However, the classical compaction of memory space by BDDs is not ideal here, since our integer variables would need to be encoded by booleans. Therefore, we use Sharing Trees instead, a compact data structure that can accommodate arbitrary data types. Preliminary results on this implementation are reported.

1 Introduction

Computer systems are now pervasive, i.e., used in all domains of human activity. In most cases, these systems, e.g.: nuclear plant controllers, airplane piloting systems, train security systems, medical monitoring computers are:

critical, i.e., errors in such systems are not acceptable due to the potentially dramatical consequences in terms of human lives, human health or financial losses;

distributed, i.e., implemented by several independent computers. This implies that the traditional testing "method" is hopeless, due to the combinatorial complexity of concurrent executions;

reactive, i.e., the evolution of these systems over time is more important than their final result, making the classical alternative to testing, namely the proof by pre- and post-conditions, inapplicable.

real-time, i.e., the timely delivery of results is as important as the results themselves. This renders the recently developed techniques of temporal logic [25,27] incomplete, and has thus triggered foundational research to add real-time to temporal formalisms [1,5,2,3,6,19].

We see thus a large gap between the strong social demand for safe, reliable computer systems and the weak methods now used to build them. We are thus investigating methods to develop real-time systems, at the foundational [31,19,29,34], methodological and practical level.

In this paper we relate the foundational level of deciding real-time logics with the practical aspects of their implementation. We do not talk about symbolic model *checking* (SMC), which is only applicable when the system has been implemented to the point where a model can be extracted from the implementation, but about symbolic model *generation* (SMG), a germane technique that allows to work on systems described by logical formulae, thus already applicable in the early stages of real-time software development, such as requirements specification and architectural specification. This technique is well-known for linear temporal logic [37]: given a logical formula, it builds an automaton that accepts exactly the models of this formula. The relations between formulae (entailment, equivalence, etc.) are then decided by operations on automata. A special kind of automata is needed for each type of logic: for instance, [19] discovered the automata corresponding to $\mathsf{MITL}^{\mathsf{DT}}$. However, using non-classical automata has its costs in terms of implementation, since baroque data structures are often needed. Thus some researchers [10] have proposed to model real-time by a state counter, a fairly counter-intuitive model that has the advantage of an immediate implementation in known temporal techniques. Here we propose a median way. We use dense time at the specification level, due to its naturalness and semantical advantages, including compositionality, ease of temporal refinement, etc.[7].

The resulting logic is undecidable, but for all practical purposes can be approximated by its fictitious clock counterpart presented in Section 3. The fictitious clock model approximates real-time, intuitively, by considering that a perfect discrete clock is started together with the system. All times are read on the integer clock display, which is a truncation of the real time. The clock display is incremented at clock ticks. Therefore the time of events is only known up to one tick delay. This imprecision makes the logic discrete and decidable by classical techniques (adapted in Section 4), while the error can be made as small as wanted by increasing the tick rate.

Looking into the technique, we see that the introduction of prophecy variables (Subsection 4.3) improves the efficiency of the procedure. We can design a logic (called $\mathsf{ECL}^{\mathsf{FC}}$) from these variables (Subsection 4.4), and as expected this logic is more efficient, but surprisingly perhaps keeps the same expressive power in this discrete model. In [33], we show that in contrast expressive power is lost in the pointwise model.

In Section 5, we consider how to implement the decision procedure. Since it is complete for memory space, we must use memory compression techniques. The classical technique is ordered binary decision diagrams (OBDDs) [8], but it has the drawback of accommodating only booleans. It is still usable at the price of an encoding of integer prophecy variables by booleans. However, the related technique of Sharing Trees (STs) [41,39] allows to accommodate directly integers and has otherwise similar performances for the needs of model generation. When dealing with state-by-state exploration [17,26], it is even much more efficient. The logical operations translate simply on this data structure. We conclude by reporting preliminary experiments on our implementation, and plans for further experiments.

2 Real-time specification

Our approach to real-time, distributed, reactive systems is currently centered around the specification language Albert (Agent-oriented Language for Building and Eliciting Requirements for Real-Time) [15,14,32]. It is built on basis of the logic MTL [23,21,22,6], extended with first-order, structuring, actions with duration, and communication concepts, and with a number of *patterns* that allow a direct representation of common sentences without the need to delve into the joys of nested untils. The definition of Albert [11,13] is too large to be presented here.

MTL is an extension of linear temporal logic, where the temporal operator is extended by a constraint on the distance at which the event will occur.

MTL can be given a number of semantics. [22] just gives basic requirements on the domain of time. [5] use the real numbers as time domain, and prove that the logic is undecidable in this case. Nevertheless, we used this latter logic as the basis of Albert [15], since the extension to first-order renders the logic undecidable anyway, and is needed for most practical applications. We consider thus decidability as a problem for tools: tools that require some form of decidability will select an adequate fragment of the language and perform an abstract interpretation [12] to get rid in a meaningful way of the specification parts that they cannot deal with. Even when the specification logic is theoretically decidable, this approach has to be followed to obtain results in a reasonable amount of time. Conversely, some tools do not require decidability and work satisfactorily in practice [18]. This is thus also the approach of this paper: we present a logic $\mathsf{MTL}^{\mathsf{FC}}$ that approximates MTL. The theory is established in [29]. MTL is itself an approximation of Albert by using known techniques to simplify the first-order part of Albert. In subsection 4.5, we approximate $\mathsf{MTL}^{\mathsf{FC}}$ by $\mathsf{ECL}^{\mathsf{FC}}$ for efficiency.

3 Fictitious clock

Given a real-time temporal logic where time is represented by real numbers, we can always define its fictitious clock (FC) counterpart. In the semantics, wherever a time was used, we replace it by a natural number that is the truncated value of the real time, for which we use the metaphor of the *clock display*. In the syntax, we add a special proposition tick that indicates when the FC will tick. Ticks are exactly separated by a distance of one *tick delay*, a parameter that is set freely by the user. A small tick delay will give rise to more precise, but more complex computations. The *tick rate* is the inverse of the tick delay. Each element of a life (model) will describe an constant state of affairs, in which neither the FC nor the system evolves. It can thus never last more than one tick delay. The relation between the two logics is thus established at the levels of models. For all practical purposes, however, we need a relation between formulae, which is deduced by the theory of abstract interpretation [12]. Thus we compute recursively the best upper approximation and the best lower approximation of each real valued formula: roughly, the lower approximation implies the real valued formula, which implies its upper approximation. For instance, if we want to check whether a real valued formula is satisfiable, we compute its lower and upper approximations and check their satisfiability in the decidable FC logic. If the lower approximation is satisfiable, then so is the real valued formula. If the upper approximation is not satisfiable, then so is the real valued formula. It is also possible that none of the above cases applies, in which case the real-valued formula is undecided, and we recommend to start again with a faster FC. The computation performed with the slower FC can be reused, see [29] for the exact conditions and the complete theory.

In the specific case of MTL, the resulting logic is thus $\mathsf{MTL}^{\mathsf{FC}}$:

Definition 3.1 [$\mathsf{MTL}^{\mathsf{FC}}$-syntax] A formula ϕ of $\mathsf{MTL}^{\mathsf{FC}}$ abides the following syntax:

$$\phi ::= p \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \phi_1 \mathcal{U}_{\sim c} \phi_2 \mid \bigcirc \phi \mid \mathsf{tick}$$

where:

- ϕ, ϕ_1, ϕ_2 are recursively formulae of $\mathsf{MTL}^{\mathsf{FC}}$,
- $\sim$ is a constraint taken from $\{<, \geq, =\}$.
- c is a rational constant,

We use the usual temporal abbreviations and precedences [28, page 49], [24]. *Inter alia*, $\phi \mathcal{U} \psi$ abbreviates $\phi \mathcal{U}_{\geq 0} \psi$, $\Diamond_{\sim c}\phi$ abbreviates $\top \mathcal{U}_{\sim c} \phi$. Note that since the distance is always a natural value, we can define $\leq c$ as $< c+1$ and $> c$ as $\geq c+1$, which would not be possible for real time.

The semantics of $\mathsf{MTL}^{\mathsf{FC}}$ is based on infinite timed sequence of states. Timed sequences of states are sequences of pairs composed of a state (as in temporal logic) and a time, which is a natural number for a FC logic. Thus such a pair actually describes an interval of real time during which nothing changes (neither the system nor the FC). Each change triggers a new state, and tick is true in the previous state when the FC changes.

Definition 3.2 A (FC) timing sequence is an infinite sequence $(t_i)_{i\in\mathbb{N}}$ of natural numbers where:

- $t_i \in \mathbb{N}$
- $t_0 = 0$, initially time is zero;
- $t_{i+1} = t_i$ or $t_{i+1} = t_i + 1$, time advances of one time unit or stays constant;
- $\forall t \exists i : t_i > t$, time never stops.

Definition 3.3 A (FC) timed trace is an infinite sequence $\tau = ((s_i, t_i))_{i\in\mathbb{N}}$

- $s_i \subseteq \mathcal{P}$, that is each s_i is the subset of propositions that are true in the i^{th} observation of the trace τ;
- $(t_i)_{i\in\mathbb{N}}$ is a FC timing sequence, giving the FC displays.

Definition 3.4 [$\mathsf{MTL}^{\mathsf{FC}}$ semantics] A $\mathsf{MTL}^{\mathsf{FC}}$ formula holds in τ at position i, noted $(\tau, i) \models \phi$, iff:

(i) $(\tau, i) \models p$ iff $p \in s_i$ for $p \in \mathcal{P}$.
(ii) $(\tau, i) \models \phi_1 \vee \phi_2$ iff $(\tau, i) \models \phi_1$ or $(\tau, i) \models \phi_2$.
(iii) $(\tau, i) \models \neg\phi$ iff not $(\tau, i) \models \phi$.
(iv) $(\tau, i) \models \bigcirc\phi$ iff $(\tau, i+1) \models \phi$.
(v) $(\tau, i) \models \phi_1 \mathcal{U}_{\sim c} \phi_2$ iff $\exists k \geq i$ such that $(\tau, k) \models \phi_2$ and $t_k - t_i \sim c$ and $\forall j : i \leq j < k : (\tau, j) \models \phi_1$.
(vi) $(\tau, i) \models \mathsf{tick}$ iff $t_{i+1} = t_i + 1$.

Example 3.5 [$\mathsf{MTL}^{\mathsf{FC}}$ formulae]

- $\Box(p \rightarrow \Diamond_{\leq 5} q)$: a p position is always followed by a q position before the FC ticks 6 times, ensuring thus a real delay strictly less than 6 tick delays. Such a formula allows the specification of bounded response time, though the actual bound is slightly unexpected.
- $\Box(p \rightarrow \Diamond_{=3} q)$: a p position is always followed by a q position 3 ticks later. Since the FC cannot identify precisely the time of p and q, the real delay is between 2 and 4.

This logic is thus somewhat imprecise, but this is necessary to ensure decidability (see also [4] for a different but similarly motivated logic). In the next section, we show decidability by giving practical algorithms.

4 A decision procedure for $\mathsf{MTL}^{\mathsf{FC}}$

The real-time logics are often undecidable, and even when they are decidable [29,19,4] their decision procedures requires non-classical data structures, e.g. open/closed hyperpolyhedra. The FC logics have a much more classical structure than their real-time counterpart: all techniques from usual temporal logic apply directly. Below, we present the classical tableau technique adapted to the case of $\mathsf{MTL}^{\mathsf{FC}}$. Let us recall that the tableau technique will build a finite automaton that recognize exactly the models of the original formula. To this purpose, it selects the formulae relevant to the formula to be decided (essentially its sub-formulae) and considers all possible combinations and all possible transitions between them, according to the semantics of the logic.

4.1 Fictitious Clock Timed Automata

Definition 4.1 [$\mathsf{TA}^{\mathsf{FC}}$] A FC timed automaton A is composed of:
(i) Q, a finite set of *states*;
(ii) $Q_0 \subseteq Q$, the *initial* states;
(iii) $E \subseteq Q \times Q$ is the *transition relation*;
(iv) $\lambda : Q \to \mathcal{P} \cup \{\mathsf{tick}\}$ is a *labelling* function that labels each location with a subset of the propositions and the special proposition tick;
(v) $Q_F \subseteq 2^{2^Q}$ is a generalized Büchi *acceptance* condition (defined below).

Note that we do not introduce FCs in the automata, but a special proposition tick. FCs would give more succinct automata, with the same expressive power.

The (operational) semantics of these automata is given below:

Definition 4.2 A timed run of a FC automaton A is an infinite sequence of pairs $\sigma = (q_0, t_0)(q_1, t_1) \dots (q_n, t_n) \dots$ where:
(i) $q_i \in Q$;
(ii) $t_0, t_1, \dots, t_n, \dots$ is a timing sequence;
(iii) $q_0 \in Q_0$: the first state is an initial state of A;
(iv) $(q_i, q_{i+1}) \in E$: the transition rule of A is respected;
(v) $\mathsf{tick} \in \lambda(q_i)$ iff $t_{i+1} = t_i + 1$, i.e., the FC ticks as noted in A.
Furthermore, the run σ is *accepted* iff for every set $F \in Q_F$ there exists infinitely many positions i such that $q_i \in F$.

Definition 4.3 [Trace of a run] The trace $\tau = (s_0, t_0)(s_1, t_1) \dots (s_n, t_n) \dots$ of a timed run $\sigma = (q_0, t_0)(q_1, t_1) \dots (q_n, t_n) \dots$ is given by $s_i = \lambda(q_i) \cap \mathcal{P}$.

Theorem 4.4 *The emptiness problem for $\mathsf{TA}^{\mathsf{FC}}$, i.e., deciding whether there exists an accepted trace, is* NLogSpace-Complete.

Note that if we would have introduce discretely valuated clocks, the emptiness problem would have been PSpace (as automata with clocks are exponentially more succinct).

4.2 Tableau Procedure

We'll show that, using a classical tableau method, for every formula ϕ of the logic $\mathsf{MTL}^{\mathsf{FC}}$ we can construct a $\mathsf{TA}^{\mathsf{FC}}$ A_ϕ that accepts exactly the models of ϕ. In the next subsection, we decide the satisfiability of $\mathsf{MTL}^{\mathsf{FC}}$ by solving the emptiness problem for $\mathsf{TA}^{\mathsf{FC}}$.

The basic sub-formulae will give the information to keep in a state:

Definition 4.5 [Basic sub-formulae] The set of basic sub-formulae of a $\mathsf{MTL}^{\mathsf{FC}}$ formula ϕ is:

- $\mathsf{basic}(p) = \{p\}$ if $p \in \mathcal{P}$;
- $\mathsf{basic}(\neg p) = \mathsf{basic}(p)$;
- $\mathsf{basic}(\psi_1 \vee \psi_2) = \mathsf{basic}(\psi_1) \cup \mathsf{basic}(\psi_2)$;
- $\mathsf{basic}(\bigcirc\psi) = \mathsf{basic}(\psi) \cup \{\bigcirc\psi\}$;
- $\mathsf{basic}(\psi_1 \mathcal{U}_{\sim c}\psi_2) = \mathsf{basic}(\psi_1) \cup \mathsf{basic}(\psi_2) \cup_{0\leq i\leq c} \{\psi_1 \mathcal{U}_{\sim i}\psi_2\} \cup \{\mathsf{tick}\}$.

In the sequel, we'll write $\phi \in q$ also for boolean combinations of basic formulae: the boolean components of ϕ are first evaluated according to the boolean rules, and only then basic formulae are checked.

The following equivalences show how the requirement expressed by an until formula can be decomposed into a requirement on the present state and a requirement on the following state. Those equivalences will be used to define the transition relation of the automaton A_ϕ for the formula ϕ.

Definition 4.6 [Expansion Formulae] Let p be a positive natural number.

(i) (a) $\psi_1 \mathcal{U}_{<p}\psi_2 \equiv \psi_1 \mathcal{U}_{\leq p-1}\psi_2$
   (b) $\psi_1 \mathcal{U}_{<0}\psi_2 \equiv \bot$.
(ii) (a) $\psi_1 \mathcal{U}_{=p}\psi_2 \equiv \psi_1 \wedge (tick \rightarrow \bigcirc(\psi_1 \mathcal{U}_{=p-1}\psi_2)) \wedge (\neg tick \rightarrow \bigcirc(\psi_1 \mathcal{U}_{=p}\psi_2))$
   (b) $\psi_1 \mathcal{U}_{=0}\psi_2 \equiv \psi_2 \vee (\psi_1 \wedge \neg tick \wedge \bigcirc(\psi_1 \mathcal{U}_{=0}\psi_2))$
(iii) (a) $\psi_1 \mathcal{U}_{\geq p}\psi_2 \equiv \psi_1 \wedge (tick \rightarrow \bigcirc(\psi_1 \mathcal{U}_{\geq p-1}\psi_2)) \wedge (\neg tick \rightarrow \bigcirc(\psi_1 \mathcal{U}_{\geq p}\psi_2))$
   (b) $\psi_1 \mathcal{U}_{\geq 0}\psi_2 \equiv \psi_2 \vee (\psi_1 \wedge \bigcirc(\psi_1 \mathcal{U}_{\geq 0}\psi_2)))$

We are now equipped to define the FC timed automaton A_ϕ that accepts exactly the models of the formula ϕ:

Definition 4.7 [$\mathsf{MTL}^{\mathsf{FC}}$-$A_\phi$] A_ϕ has the following elements:

(i) the set of states Q are subsets of $\mathsf{basic}(\phi) \cup \{\mathsf{tick}\}$
(ii) the initial states contain ϕ: $Q_0 = \{q \in Q | \phi \in q\}$
(iii) the transition relation is defined according to the expansion formulae: $(q_1, q_2) \in E$ iff for every $\psi \in \mathsf{basic}(\phi)$, its expansion rule is verified when the $\bigcirc$-formulae appearing in the expansion are evaluated in q_2. For example, if $(q_1, q_2) \in E$ and $\psi_1 \mathcal{U}_{\leq 2}\psi_2, \psi_1, \mathsf{tick} \in q_1$ then the expansion formulae of definition 4.6 tell us that $\psi_1 \mathcal{U}_{\leq 1}\psi_2 \in q_2$.
(iv) the labelling function is obvious: $\lambda(q) = q \cap (\mathcal{P} \cup \{\mathsf{tick}\})$;
(v) the set of accepting locations Q_F contains the sets:
   (a) $F_{\mathsf{tick}} = \{q \mid \mathsf{tick} \in q\}$; this set ensures that time goes beyond any bounds, i.e. that the FC ticks infinitely often.
   (b) for every formula g of the form $\psi_1 \mathcal{U}_{\sim c}\psi_2 \in \mathsf{basic}(\phi)$, with $\sim \in \{>, \geq\}$, a set $F_g = \{q \mid \psi_2 \in q \vee g \notin q\}$

Theorem 4.8 *The FC timed traces that are accepted by the automaton A_ϕ are exactly the models of ϕ.*

4.3 Prophecy Variables

In the construction given in definition 4.7, $c+1$ subformulae are introduced for each formula of the form $\psi_1\,\mathcal{U}_{\sim c}\psi_2$. So each such formula multiply by 2^{c+1} the number of states in A_ϕ. This explosion is not necessary if $\sim \in \{<, \leq, \geq, >\}$. For instance, let us examine the formula $\Psi_c \equiv \psi_1\,\mathcal{U}_{\leq c}\psi_2$. If this formula is true in position i of timed trace τ then the formulae Ψ_d with $0 \leq d < c$ are also true in that position i. So instead of maintaining the truth value of all those Ψ_d, with $0 \leq d \leq c$, we simply have to remember the time distance e that separates the position i from the first instant when ψ_2 will be true. Then we can reconstruct all the information: If $\psi_1\,\mathcal{U}\psi_2$ is currently true then for all d such that $0 \leq d < e$, Ψ_d is false and for all d such that $d \geq e$, Ψ_d is true. Otherwise, all $\mathcal{U}$ are false. e is called the value of the *prophecy variable* for ψ_2.

4.4 Event Clock Logic

This simple idea is captured by the logic of event clocks:

Definition 4.9 [$\mathsf{ECL}^{\mathsf{FC}}$-syntax] The grammar of $\mathsf{ECL}^{\mathsf{FC}}$ is:

$$\phi ::= p \mid \phi_1 \vee \phi_2 \mid \bigcirc \phi_1 \mid \phi_1\mathcal{U}\phi_2 \mid y_{\phi_1} \sim c \mid \mathsf{tick}$$

Definition 4.10 [$\mathsf{ECL}^{\mathsf{FC}}$-semantics] The semantics is as for $\mathsf{MTL}^{\mathsf{FC}}$, except (of course) for prophecy variables. The *value* of y_ϕ in τ at i, noted $\mathsf{val}_{(\tau,i)}(y_\phi)$, is:
$\mathsf{val}_{(\tau,i)}(y_\phi) = t_j - t_i$ iff $j \geq i$, $(\tau, j) \models \phi$, and $\forall k$, $i \leq k < j$, $(\tau, j) \not\models \phi$;
There might be no such j, in which case the value is undefined, and any constraint on an undefined value is conventionally false. The semantics of the prophecy formulae is now obvious:
$(\tau, i) \models y_\phi \sim c$ iff $\mathsf{val}_{(\tau,i)}(y_\phi) \sim c$

We now redefine an optimised notion of "basic" for $\mathsf{ECL}^{\mathsf{FC}}$:

Definition 4.11 [basic $\mathsf{ECL}^{\mathsf{FC}}$ formulae and variables] The basic sub-formulae of a $\mathsf{ECL}^{\mathsf{FC}}$-formula ϕ are defined by:
(i) $\mathsf{basic}(p) = \{p\}$ if $p \in \mathcal{P}$;
(ii) $\mathsf{basic}(\neg\psi) = \mathsf{basic}(\psi)$;
(iii) $\mathsf{basic}(\psi_1 \vee \psi_2) = \mathsf{basic}(\psi_1) \cup \mathsf{basic}(\psi_2)$;
(iv) $\mathsf{basic}(\bigcirc\psi) = \mathsf{basic}(\psi) \cup \{\bigcirc\psi\}$;
(v) $\mathsf{basic}(\psi_1\mathcal{U}\psi_2) = \mathsf{basic}(\psi_1) \cup \mathsf{basic}(\psi_2) \cup \{\psi_1\mathcal{U}\psi_2\}$;
(vi) $\mathsf{basic}(y_\psi \sim c) = \{y_\psi \sim c\} \cup \{\mathsf{tick}\} \cup \mathsf{basic}(\psi)$;
From the basic formulae of the form $y_\psi \sim c$, we extract the *basic variable* y_ψ, and its *maximum* m_ψ which is the maximum of all c to which y_ψ is compared. Its *range* is $\{0, 1, \ldots, m_\psi, m_\psi^+, \perp\}$. m_ψ^+ is used to represent any value greater than m_ψ. $\perp$ represents the undefined value, used when no ψ occurs in the future. The value of the basic prophecy variable y_ψ determines all its basic prophecy formulae: thus there is no need to record basic prophecy formulae.

Definition 4.12 The expansion formulae for the temporal operators $\bigcirc$, $\mathcal{U}$ are as for $\mathsf{MTL}^{\mathsf{FC}}$. The expansion formulae for a prophecy variable y_ψ are:

- if $v = 0$, then $y_\psi = 0 \equiv \psi \vee (\neg\mathsf{tick} \wedge \bigcirc y_\psi = 0)$;
- if $v \in (0, m_\psi]$, then $y_\psi = v \equiv \neg\psi \wedge (\mathsf{tick} \to \bigcirc y_\psi = v - 1) \wedge (\neg\mathsf{tick} \to y_\psi = v)$;
- if $v = m_\psi^+$, then $y_\psi = v \equiv \neg\psi \wedge \bigcirc(y_\psi = m_\psi \vee y_\psi = m_\psi^+)$;
- if $v = \bot$, then $y_\psi = v \equiv \neg\psi \wedge \bigcirc y_\psi = \bot$.

It simply expresses that the display is incremented at ticks, and thus the distance between the current display and the event is then decremented.

As usual, we can define for each $\mathsf{ECL}^{\mathsf{FC}}$-formula ϕ, a FC timed automaton A_ϕ that accepts exactly the models of ϕ:

Definition 4.13 $[\mathsf{ECL}^{\mathsf{FC}} - A_\phi]$ A_ϕ has the following elements:

(i) the set of states Q are pairs (s, v) where s is a subset of the non-prophecy basic formulae, (the ones that are true now).
(ii) the initial states contain ϕ: $Q_0 = \{q \in Q | \phi \in q\}$
(iii) the transition relation is defined from the expansion, as usual; For example, if (s_1, v_1) is such that $v_1(y_\psi) = 3$ and $\mathsf{tick} \in s_1$ then for every (s_2, v_2) such that $((s_1, v_1), (s_2, v_2)) \in E$, the expansion formulae of definition 4.12 tell us that: $v_2(y_\psi) = 2$.
(iv) the labelling function is defined by: $\lambda((s, v)) = s \cap (\mathcal{P} \cup \{\mathsf{tick}\})$;
(v) the set of accepting locations Q_F contains:
   (a) for every formula g of the form $\psi_1 \mathcal{U} \psi_2$ or $y_\psi > m_\psi \in \mathsf{basic}(\phi)$ (coded as $y_\psi = m_\psi^+$), a set $F_g = \{q \mid \psi \in q \vee g \notin q\}$; each promising formula g is followed later by its promised formula ψ;
   (b) a set $F_{\mathsf{tick}} = \{q \mid \mathsf{tick} \in q\}$; time goes beyond any bounds.

Note that although $y_\psi = c$ is a promise for ψ, we do not need to include specific accepting locations, since the ticks will eventually bring about ψ.

4.5 Using $\mathsf{ECL}^{\mathsf{FC}}$ to decide $\mathsf{MTL}^{\mathsf{FC}}$

Most operators of $\mathsf{MTL}^{\mathsf{FC}}$ translate efficiently into $\mathsf{ECL}^{\mathsf{FC}}$:

(i) $\psi_1 \mathcal{U}_{<c} \psi_2 \equiv \psi_1 \mathcal{U} \psi_2 \wedge y_{\psi_2} < c$
(ii) $\psi_1 \mathcal{U}_{\leq c} \psi_2 \equiv \psi_1 \mathcal{U} \psi_2 \wedge y_{\psi_2} \leq c$
(iii) $\psi_1 \mathcal{U}_{\geq c} \psi_2 \equiv \Box_{<c}(\psi_1 \wedge \bigcirc(\psi_1 \mathcal{U} \psi_2))$
(iv) $\psi_1 \mathcal{U}_{>c} \psi_2 \equiv \Box_{\leq c}(\psi_1 \wedge \bigcirc(\psi_1 \mathcal{U} \psi_2))$

In contrast, the translation of the equality is necessarily exponential, as expected from the fact that $\mathsf{MTL}^{\mathsf{FC}}$ is complete in ExpSpace, while $\mathsf{ECL}^{\mathsf{FC}}$is only complete in PSpace. A possible translation:

(i) $\psi_1 \mathcal{U}_{=0} \psi_2 \equiv (\psi_1 \wedge \neg\mathsf{tick}) \mathcal{U} \psi_2$
(ii) $\psi_1 \mathcal{U}_{=p} \psi_2 \equiv (\psi_1 \wedge \neg\mathsf{tick}) \mathcal{U}(\mathsf{tick} \wedge \psi_1 \wedge \bigcirc(\psi_1 \mathcal{U}_{=p-1} \psi_2))$

Here one can see that the source of the exponential is simply the translation of the (binary) constant p into some form of unary notation.

Using our abstract interpretation framework [29], it is possible also to use approximations. By doing so, we gain efficiency but loose precision in solving $\mathsf{MTL}^{\mathsf{FC}}$. Anyway, in our approach, $\mathsf{MTL}^{\mathsf{FC}}$ is already an approximation of Albert. For instance we can use the following linear translation:

(i) $\downarrow(\psi_1 \mathcal{U}_{=c} \psi_2) \equiv \psi_1 \mathcal{U} \psi_2 \wedge y_{\psi_2} = c$

(ii) $\uparrow(\psi_1 \mathcal{U}_{=c} \psi_2) \equiv \psi_1 \mathcal{U}_{\leq c} \psi_2 \wedge \psi_1 \mathcal{U}_{\geq c} \psi_2$

5 Sharing trees

5.1 Introduction

Constructing explicitly the automata presented is clearly not efficient enough to be implemented, since the number of their states is exponential in the formula. So in this section, we present a so called *symbolic* procedure that permits to implement efficiently the tableau procedure. The procedure is symbolic in the sense that neither individual states nor pairs of states of the transition relation of the automaton are explicitly manipulated. We use Sharing Trees (STs) [41], a data structure to represent compactly sets of tuples. For the definition of STs and a comparison with BDDs, see [39]. The main idea is to ensure *suffix merging* of tuples that share equal ends and *prefix merging* of tuples that share equal beginnings. STs are very close to minimized deterministic finite state automata without loops.

A set of operations are available over STs to efficiently manipulate set of tuples. Here we will use the following operations:

(i) Union: $ST1 \cup ST2 = \{x \mid x \in ST1 \vee x \in ST2\}$

(ii) Intersection: $ST1 \cap ST2 = \{x \mid x \in ST1 \wedge x \in ST2\}$

(iii) Matching: $Match(ST1, i, a) = \{(x_1, ..., x_n) \mid (x_1, ..., x_n) \in ST1 \wedge x_i = a\}$

(iv) Projection $\downarrow_{\bar{x}} ST = \{\bar{x} \mid \exists \bar{y} : \bar{y} \in ST \wedge var(\bar{x}) \subseteq var(\bar{y}) \wedge \bar{x} =_{var(\bar{x})} \bar{y}\}$ [1]

(v) Empty: $Empty = \emptyset$.

Those operators are implemented by symbolic algorithms in the sense that tuples are never explicitly handled but only global operations on the ST are performed. In this way, the improvement in memory space also yields an improvement in execution time, as for BDDs. However, operations such as matching and projection are more efficient. Applications of STs to represent automata can be found in [40,26].

5.2 Using Sharing Trees in $\mathsf{ECL}^{\mathsf{FC}}$ Model-Generation

In the following, we use STs to represent symbolically the automaton of a $\mathsf{ECL}^{\mathsf{FC}}$ formula. The set of states of the automaton can be represented by a

[1] $var(\bar{x})$ returns the set of variables of the vector $\bar{x}$, $\bar{x} =_{var(\bar{x})} \bar{y}$ expresses the fact that $\bar{x}$ and $\bar{y}$ give the same value to common variables.

ST that contains tuples which are a valuation of the basic formulae (except prophecy formulae) and variables of ϕ.

Given the finite domains of tuples – each element x_i either boolean, for basic non-prophecy formulae, or in $\{\bot, 0, \dots, m_\psi, m_\psi+\}$ for prophecy formulae – we can add basic operations:

(i) True: $FullST$ is the ST that contains all possible combinations.

(ii) Complement: $Compl(ST1) = \{x \mid x \notin ST1\}$. It represents the negation.

(iii) $Form(ST, i)$ returns the formula of the i^{th} layer of ST.

(iv) $Match(ST, \phi)$ abbreviates $Match(ST, i, 1)$ where $Form(ST, i) = \phi$.

Now we define the symbolic representation by STs of the automaton $A_\phi = (Q, Q_0, T, \lambda, \mathcal{F})$ of a $\mathsf{ECL}^{\mathsf{FC}}$ formula ϕ.

The set of **states** Q can simply be represented by $FullST$. Sometimes, we will use the set of reachable states, the set of states that can reach an accepting state, the locally consistent states (i.e. the states that can have transitions) instead, for efficiency reasons. The other sets will then be intersected with Q, or – more efficiently – we can use Q as a relevance set, allowing the implementation to choose the most compact representation that preserves the intersection with Q.

Let us now see how to compute symbolically [2] the set of **initial states** Q_0 of A_ϕ. The function ST does this by recursion on the formula:

Definition 5.1 [Sharing Tree of a $\mathsf{ECL}^{\mathsf{FC}}$ formula] The ST of a $\mathsf{ECL}^{\mathsf{FC}}$ formula ϕ contains the states that verify ϕ.

- if $\phi \in basic(\phi), \phi \neq y_\psi \sim c$: $ST(\phi) = Match(FullST, \phi)$
- if $\phi = y_\psi \sim c$: $ST(\phi) = \bigcup_{d \sim c} Match(FullST, y_\psi, d)$.
- $ST(\neg\phi) = Compl(ST(\phi))$
- $ST(\phi \vee \psi) = ST(\phi) \cup ST(\psi)$
- $ST(\phi \wedge \psi) = ST(\phi) \cap ST(\psi)$

Note that modal formulae are always basic, so that they are already considered in this case analysis.

The **transition relation**, a set of pairs of states, will be represented by a ST with two alternated sets of variables, representing respectively the old and the new state, as is classical for BDDs, in order to take advantage of the direct dependencies between the old and new values of a variable.

There is no need to represent the **labelling** λ, which is fixed.

Finally, we represent symbolically the sets of **accepting** states simply by the corresponding STs, as for Q_0.

[2] We use "compute symbolically" because the ST will be obtained only by performing global operation on STs and no direct manipulations of the tuples that represents atoms

Algorithm 1. Transition relation

```
$T^{x,x'} := Q \times Q$;
$\forall \psi \in basic(\phi)$ do
    $T^{x,x'} := T^{x,x'} \cap ST(Exp(\psi, x, x'))$
```

In the actual implementation $Exp(\psi, x, x')$ is not a function, but the expansions of all possible ψ are pre-computed. Subformulae beginning with $\bigcirc$ in the expansion law are evaluated in x'.

5.3 Computing the emptiness of the symbolic automaton

We have shown how to construct symbolically A_ϕ. So to decide whether a formula ϕ is valid, we check the emptiness of $A_{\neg\phi}$.

There exists an algorithm [35] linear in the number of states, but since the number of states is usually exponential, this procedure is not interesting here. For efficiency, the emptiness check must be performed *symbolically.* Two slightly different approaches exist: an algorithm using the transitive closure of T and another one based on [16], that we use in practice, and present here.

The Emerson-Lei formula searches for a *reachable accepted cycle*, i.e. a cycle which passes through at least one state of each accepting set. Due to the generalised Büchi acceptance condition, this is exactly what is needed to construct an accepted infinite sequence.

Definition 5.2 [Accepted cycle] An accepted cycle in an automaton is a set of states C such that:

(i) $\forall F_i \in \mathcal{F} : C \cap F_i \neq \emptyset$; (accepted)

(ii) $\forall c\ , c_2 \in C : \exists y\ , \ldots, y_n \in C$ s. t. $(c\ , y\ ) \in T \wedge \ldots (y_n, c_2) \in T$. (cycle)

Algorithm 2. Reaching states

This fix point computes the set of states that can reach states of F_c by using edges of some transition relation T_c:

```
function $CanReach(F_c, T_c)$
$S := F_c$;
repeat
    $Prec := S$;
    $S^x := S^x \cup \downarrow_x (T_c^{x,y} \cap S^y)$;
until $S = Prec$
return($S$)
```

We have implemented the two algorithms. The algorithm based on the Emerson-Lei formula has given far better results. The problem with the algorithm based on the transitive closure is that it requires the computation of STs with $3n$ layers where n is the number of basic formulae. This produces

Algorithm 3. Emptiness based on Emerson-Lei

```
$C := N$;
repeat
    $Prec := C$;
    $\forall F \in \mathcal{F}$do
        $T_c := T^{x,y} \cap C^x$;
        $F_c := F \cap C$;
        $S := CanReach(F_c, T_c)$;
        $C := C \cap S$;
until $C = Prec$
return($C = \emptyset$)
```

a blow-up of the number of nodes such that the computation stops by lack of memory.

6 Example: Refining data transmission by CSMA/CD

A refinement step consists here of two $\mathsf{ECL}^{\mathsf{FC}}$ formulae ϕ and ϕ_2 such that ϕ_2 refines ϕ iff $L(\phi_2) \subseteq L(\phi)$. Typically ϕ_2 is a more operational description of the system described by the requirements ϕ. Note that the two descriptions are logical and can thus be of a fairly high level, in contrast with model checking that needs an operational description and is thus only applicable after the development.

Example 6.1 Let us consider a system composed of two computers (M, M_2) and a single line (L) that enables the two computers to send messages to the outside. We will specify the requirements of a protocol that will allow the two computers to send messages via the line as well as a description of a solution for these requirements. The decision procedure will be used to automatically prove that the described solution respects the formulated requirements.

Let us first formulate the requirements ϕ for the protocol:

(R1.1) $\Box(\mathsf{WToSent} \rightarrow \Diamond(\mathsf{BeginToSend} \wedge \bigcirc(\mathsf{InSending}\ \mathcal{U}\mathsf{EndSending})))$
When the computer M wants to send some message (WToSent), it eventually succeeds to send the message correctly. We consider that a transmission of M is correct if it begins with a BeginToSend event and terminates with a EndSending event. Between the two events, M is in transmission state (InSending).

(R1.2) is as R1.1 but for M_2.

(R1.3) $\Box\neg((\mathsf{InSending}_2\mathcal{U}\mathsf{EndSending}_2) \wedge (\mathsf{InSending}\ \mathcal{U}\mathsf{EndSending}))$
This requirement imposes that the protocol takes into account that the

communication medium (the line) is not able to transmit 2 messages at the same time. Thus succeeding communications can not overlap each other.

Those three formulae are high level requirements for the protocol. Those requirements are highly declarative. We think that it would be helpful to derive more detailed and more operational requirements for the protocol before trying to give an operational solution, for example in term of a finite state machine, that implements the requirements.

In the following we will concentrate on the refinement of requirement $R1.3$. A possible solution to this requirement is given by the protocol CSMA/CD:

Definition 6.2 [CSMA/CD] When a computer wants to send a message, it tests whether the line is busy. If the line is not busy, it begins to send; else it waits. A *collision* occurs when more than one computer are transmitting at the same time. The delay of propagation plays an important role in the protocol: If one computer begins to send, the other computers will not immediately see that the computer is sending, but they will see it at most α later, where α is the propagation delay between the two most distant computers. Consequently, a collision may occur between 0 and α after a computer has begun to send. The noise of the collision can also take α to reach the computer which is sending. A computer is sure that no collision will occur only after 2α.

In term of the CSMA/CD, the requirement $R1.3$ can be rewritten in:

(R1.3.1) $\mathsf{CD} \equiv \mathsf{InSending}\ \land \mathsf{InSending}_2$

There is a collision when two computers are sending at the same time.

(R1.3.2) $\Box(\neg(\mathsf{InSending}\ \mathcal{U}\mathsf{EndSending}\ ) \land CD)$

(R1.3.3) $\Box(\neg(\mathsf{InSending}_2 \mathcal{U}\mathsf{EndSending}_2) \land \mathsf{CD})$

If there is a collision during a transmission, it cannot succeed.

Let us now try to refine the requirement $R1.3.2$ with the solution proposed by the $CMAS/CD$ protocol. We must thus describe a solution (or a more operational requirement) which implies the requirement $R1.3.2$:

(I.1) $\Box(\mathsf{CD} \rightarrow \Diamond_{\leq 2}\mathsf{MSeeCD}\ )$

When there is a collision, the computers see it at latest 2 time units later (we take here the propagation delay as time unit). Due to the use of the FC semantics, the formula really expresses that the collision is detected before the 3rd tick.

(I.2) $\Box(\neg\mathsf{InSending}\ \land \bigcirc\mathsf{InSending}\ \leftrightarrow \mathsf{BeginToSend}\ )$

The definition of $\mathsf{BeginToSend}$.

(I.3) $\Box(\mathsf{InSending}\ \land \bigcirc\neg\mathsf{InSending}\ \rightarrow \mathsf{EndSending}\ \lor \mathsf{MSeeCD}\ )$

If the computer stops sending, either it is because it has finished to send its message ($\mathsf{EndSending}$) or because it has detected a collision (MSeeCD).

(I.4.1) $\Box(\mathsf{EndSending}\ \rightarrow \mathsf{InSending}\ \land \bigcirc\neg\mathsf{InSending}\ )$

To stop sending, M must be sending, and just after M does not send.

(I.4.2) $\Box(\mathsf{MSeeCD}\ \rightarrow \bigcirc\neg\mathsf{InSending}\ )$

If M detects a collision, it stops to send.

(I.4.3) $\Box(\neg(\mathsf{EndSending}_1 \wedge \mathsf{MSeeCD}_1))$
A collision can not be considered as a normal end of transmission.

(I.5) $\Box(\mathsf{BeginToSend}_1 \rightarrow \Box_{\leq 4}\neg\mathsf{EndSending}_1)$
To ensure that a computer always detects a collision during one of its transmissions, the duration of the messages must be at least 2α. Due to the FC semantics we have modeled this requirement by constraining the begin and the end of a transmission to be distant of at least 5 ticks.

(I.6) $\Box(\mathsf{InSending}_1 \wedge \mathsf{CD}) \rightarrow \neg\mathsf{CD}\,\mathcal{U}\neg\mathsf{InSending}_1)$
If M_1 is in transmission state at least during the time that separates 2 ticks of the FC and no collision occurs then no collision will occur before M_1 ends to send. All other computers have seen that the line was busy.

(I.7) $\neg\mathsf{InSending}_1$
Initially, the computers do not send a message.

We must now show that I.1-I.7 is a refinement of requirement R1.3.2. We have submitted the formula (I.1-I.7) $\rightarrow$ (R1.3.2) to our decision procedure, it was proven instantaneously.

This shows that although this problem is not naturally expressed in the FC semantics, it can be translated automatically and solved there [29]. We hope to generalize these findings to equip our real-time specification language with a tool box of translators and solvers, integrated by an interactive proof system.

7 Conclusions

7.1 Summary

We have explored techniques to refine real-time requirements. We have first defined a FC real-time logic called $\mathsf{MTL}^{\mathsf{FC}}$. It is used as an abstraction of the Albertlanguage. We have shown the ability of the logic for the specification of properties of concurrent and reactive systems. We have presented a procedure to construct the automaton that accepts exactly executions corresponding to the models of a formula of $\mathsf{MTL}^{\mathsf{FC}}$ logic. The theoretical analysis showed that a slightly different logic $\mathsf{ECL}^{\mathsf{FC}}$ would be more efficient, without loss of expressivity. In practice, we use the union of the two logics, that are intertranslatable. Due to the high memory consumption foreseen by theory, this procedure has been implemented *symbolically* using STs. A small example coming from practice, which is only proved with difficulty by humans, showed the applicability of the approach.

7.2 Future work

The following problems are addressed in other papers:

- Logics with real can be decidable: [30] present a decidable continuous-time logic; [19] constructs the associated automata and monadic logics, which are also decidable. Thus we could extend our logics in the spirit of [36] without loosing decidability.

- The theory of abstract interpretation that we use is presented in [29]. We are preparing a more general version.
- The logic is propositional and has no structuring means. In [11], we define a continuous-time, structured, first-order specification language, that shares its semantic foundations with the logics presented here.
- The methodology to obtain formal requirements and its integration into existing methods are currently under investigation; see [20,38] for preliminary thoughts.

We intend to deal with the following problems in the future:

- the state explosion problem limits the formulae that can be treated automatically. So there is no hope to deal with complete problems with this type of technique alone.

  It is why we think that the decision procedure should better be used within an interactive theorem prover, equipped with the theory of abstract interpretation: the type of procedure presented here would serve as a means to abstract more expressive logics and to automate part of proofs. The abstraction can either be:

(i) automatic, as we intend to do for the FC abstraction, where a good tick rate can be obtained by doubling it until success (or memory exhaustion).

(ii) manual: for many abstractions, a proof showing the validity of the technique is required.

  Thus an interactive prover provides the glue needed to put the results of automatic techniques into a safe use. Too often, the proponents of model checking use models that have been devised manually and whose correspondence with the real system is questionable.

References

[1] R. Alur. *Techniques for Automatic Verification of Real-Time Systems.* PhD thesis, Stanford University, 1991.

[2] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993. Preliminary version appears in the Proc. of 5th LICS, 1990.

[3] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994. Preliminary version appears in Proc. 17th ICALP, 1990, LNCS 443.

[4] R. Alur, T. Feder, and T. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[5] R. Alur and T. Henzinger. Real-time logics: complexity and expressiveness. *Information and Computation*, 104(1):35–77, 1993. Preliminary version appears in the Proc. of 5th LICS, 1990.

[6] R. Alur and T. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994. Preliminary version appears in Proc. 30th FOCS, 1989.

[7] H. Barringer, R. Kuiper, and A. Pnueli. A really abstract concurrent model and its temporal logic. In *Proceedings of the 13th Annual Symposium on Principles of Programming Languages*, pages 173–183. ACM Press, 1986.

[8] R. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[9] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the 5th Symposium on Logic in Computer Science*, pages 428–439, Philadelphia, June 1990.

[10] S. Campos, E. Clarke, W. Marrero, M. Minea, and H. Hiraishi. Computing quantitative characteristics of finite-state real-time systems. In *Proceedings of the 15th Annual Real-time Systems Symposium*. IEEE Computer Society Press, 1994.

[11] F. Chabot, L. Ferier, J.-F. Raskin, and P.-Y. Schobbens. The formal semantics of Albert II. Technical report, University of Namur, 1999.

[12] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of Fourth ACM Symposium on Programming Languages (POPL'77)*, pages 238–252, Los Angeles, California, Jan. 1977.

[13] P. Du Bois. Intuitive definition of the Albert II language. Technical report, Computer Science Department, University of Namur, Namur (Belgium), February 1995.

[14] E. Dubois, P. Du Bois, and M. Petit. Albert: an Agent-oriented Language for Building and Eliciting Requirements for real-Time systems. In *Proc. of the 27th Hawaii International Conference on System Sciences – HICSS-27*, Maui (Hawaii), January 1994. IEEE.

[15] E. Dubois, P. Du Bois, and A. Rifaut. Elaborating, structuring and expressing formal requirements of composite systems. In P. Loucopoulos, editor, *Proc. of the 4th conference on advanced information systems engineering – CAiSE'92*, pages 327–347, Manchester (UK), May 12-15, 1992. LNCS 593, Springer-Verlag.

[16] E. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proc. 18th Hawaii International Conference on System Sciences*, Hawaii, 1985.

[17] F. Gagnon, J.-C. Gregoire, and D. Zampunieris. Sharing trees for "on-the-fly" verification. In *Proceedings of the 8th International IFIP Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE'95)*, Montreal (Quebec), 1995. IEEE.

[18] T. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: the next generation. In *Proceedings of the 16th Annual Real-time Systems Symposium*, pages 56–65. IEEE Computer Society Press, 1995.

[19] T. Henzinger, J.-F. Raskin, and P.-Y. Schobbens. The regular real-time languages. In K. Larsen, S. Skyum, and G. Winskel, editors, *ICALP'98: Automata, Languages, and Programming*, Lecture Notes in Computer Science 1443, pages 580–591. Springer-Verlag, 1998.

[20] P. Heymans and E. Dubois. Scenario-based techniques for supporting the elaboration and the validation of formal requirements. *Requirements Engineering Journal (to appear)*, 1999.

[21] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-time Systems*, 2(4):255–299, 1990.

[22] R. Koymans. *Specifying message passing and time-critical systems with temporal logic.* LNCS 651, Springer-Verlag, 1992.

[23] R. Koymans, J. Vytopil, and W. de Roever. Specifying message passing and time-critical systems with temporal logic. Doctoral dissertation, Eindhoven University of Technology, Eindhoven (The Netherlands), 1989.

[24] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, Berlin, January 1992.

[25] J. S. Ostroff. *Temporal Logic for Real-Time Systems.* Advances Software Development Series. Research Studies Press Ltd. (Wiley), 1989.

[26] R. Paquay and D. Zampunieris. Mec with sharing trees. Research Paper RP-96-006, Computer science department, FUNDP, Namur (Belgium), mai 1996.

[27] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symposium on Foundation of Computer Science*, pages 46–57, 1977.

[28] J.-F. Raskin. *Logics, Automata and Classical Theories for Deciding Real Time.* PhD thesis, Institut d'Informatique, FUNDP, Namur, June 1999.

[29] J.-F. Raskin and P.-Y. Schobbens. Real-time logics: Fictitious clock as an abstraction of dense time. In *Proc. Third International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS97)*, volume 1217 of *Lecture Notes in Computer Science (LNCS)*, pages 165–182. Springer, 1997.

[30] J.-F. Raskin and P.-Y. Schobbens. State clock logic : a decidable real-time logic. In *Proc. International Workshop on Real-time and Hybrid Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science (LNCS)*, pages 33–47. Springer, 1997.

[31] J.-F. Raskin, P.-Y. Schobbens, and T. Henzinger. Axioms for real-time logics. In D. Sangiorgi and R. de Simone, editors, *Proceedings of CONCUR'98: 9th International Conference on Concurrency Theory*, volume 1466 of *Lecture Notes in Computer Science (LNCS).* Springer, 1998.

[32] P.-Y. Schobbens. Albert at the age of five. In *Object-Oriented Software Development*, number 174 in Dagstuhl Seminar Bericht, April 1997.

[33] P.-Y. Schobbens and J.-F. Raskin. The logic of event clocks. *Journal of Algebra, Languages and Combinatorics*, 1999. To appear.

[34] P.-Y. Schobbens, J.-F. Raskin, and T. A. Henzinger. Axioms for real-time logics. *Theoretical Computer Science*, 1999. Submitted.

[35] R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.

[36] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, 1983.

[37] P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, (110–111):119–136, 1985.

[38] E. Yu, P. Du Bois, E. Dubois, and J. Mylopoulos. From organization models to system requirements - a "cooperating agents" approach. In *Proc. of the Third International Conference on Cooperative Information Systems – CoopIS-95*, Vienna (Austria), May 9-12, 1995. University of Toronto Press inc.

[39] D. Zampuniéris. *The Sharing Tree Data Structure: Theory and Applications in Formal Verification*. PhD thesis, Institut d'Informatique, FUNDP, Namur, May 1997.

[40] D. Zampuniéris and B. Le Charlier. An efficient algorithm to compute the synchronized product. In *Proceedings of Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'95)*, Durham, North Carolina (USA), Jan. 1995. IEEE.

[41] D. Zampuniéris and B. Le Charlier. Efficient handling of large sets of tuples with sharing trees. In *Proceedings of the Data Compression Conference (DCC'95)*, Snowbird, Utah (USA), Mar. 1995. IEEE.