Model Checking of Real-Time Reachability Properties Using Abstractions*

Conrado Daws and Stavros Tripakis

Abstract. Practical real-time model checking suffers from the *state-explosion* problem: the size of the state space grows exponentially with many system parameters: number of clocks, size of constants, number of system components. To cope with state explosion, we propose to use *abstractions* reducing the state-space while preserving *reachability properties*. Four *exact*, plus one *safe* abstractions are defined. In the main abstraction (*simulation*) a concrete state is mapped to a *symbolic* abstract state (a set of concrete states). The other four abstractions are defined on top of the simulation one. They can be computed on-the-fly in a completely orthogonal manner and thus can be combined to yield better reductions. A prototype implementation in the tool KRONOS has permitted to verify two benchmark examples with a significant scale-up in size.

1 Introduction

Model checking is an approach commonly used for the automatic verification of *reachability properties*. Given a system and a property p, reachability model checking is based on an exhaustive exploration of the reachable state space of the system, testing whether there exists a state where p holds. The main obstacle to this approach is the so-called *state-explosion problem* reflecting the fact that the system's state space is often prohibitively large to be entirely explored.

Abstractions [6, 19] have been proven a useful tool in coping with state explosion. Model checking using abstractions consists in exploring a (hopefully much smaller) *abstract* state space rather than the *concrete* one. Since the abstract space contains less information than the concrete, a crucial question is which properties are preserved by the abstraction. *Exact* abstractions imply no information loss: the abstract system satisfies a property iff the concrete one does. *Safe* abstractions ensure only one direction, that is, if a property holds on the concrete system, then it holds also on the abstract one, otherwise no definite conclusion can be made.

In the context of real-time systems modeled as *timed automata*, the state space is infinite, due to continuous variables called *clocks*, used to measure time. An abstraction is provided in [2] which is exact with respect to all properties that can be expressed in the real-time logic TCTL and which also induces a finite

* VERIMAG, Centre Équation, 2 avenue de Vignate, 38610 Gières, France. Fax. +33 4 76 63 48 50, e-mail: {Conrado.Daws, Stavros.Tripakis}@imag.fr, internet: http://www.imag.fr/VERIMAG/PEOPLE/Conrado.Daws.

abstract state space called a *region graph*. Unfortunately, the size of the latter is exponential on the number of clocks and on the size of the constants against which the clocks are compared to, in the automaton in question. Therefore, the applicability of region-graph-based approaches remains very limited in practice.

In this paper, we propose five coarser abstractions in order to cope with state-explosion. Four of these are exact with respect to reachability, while the fifth one is safe. The main abstraction is defined on the infinite concrete state space, and is based on the concept of *simulation space*, where abstract states are *symbolic*, that is, predicates characterizing sets of concrete states. The simulation space is obtained as the fix-point of a *successor* (or *post-condition*) operator on symbolic states. The remaining four abstractions are defined on the simulation space, that is, both concrete and abstract states are symbolic. The *extrapolation* abstraction is needed to ensure that the simulation space is finite. The *inclusion* abstraction reduces the number of symbolic states by mapping subsets of concrete states to the same abstract state. The *activity* abstraction reduces the number of clocks by eliminating those which are not active at some point during the exploration. Finally, the *convex-hull* abstraction collapses symbolic states which are associated with the same control location to a single abstract state, the clock part of which is the convex hull of the clock parts of the concrete states. All abstractions are exact, except for the convex-hull one, which is safe.

An important feature of these four abstractions is that they are completely *orthogonal* to each other, that is, they can be composed to yield abstractions which are more powerful in terms of state-space reduction. This results in no loss of information, since the composition of exact abstractions is also exact, while the composition of exact and safe abstractions is safe. Section 3 contains the definitions of the abstractions, as well as some examples.

Section 4 presents our model-checking approach, which consists in generating and exploring *on-the-fly* an abstract state space [1]. This is done using a depth-first or breadth-first search in which the successor operator and storage procedures are parameterized by the abstraction(s) applied.

A prototype implementation of these features has been done on top of the real-time-verification tool KRONOS. Experimental results obtained on two benchmark examples are presented in section 5. Using abstractions, we have been able to verify these examples for a much larger number of components, with respect to previous attempts using explorative-model-checking tools like KRONOS, UPPAAL, HYTECH or RTSPIN. Section 5.2 also compares the results to explorative techniques based on binary decision diagrams (BDDs). Conclusions are presented in section 6.

[1] The term on-the-fly is taken to mean two things in this paper: (a) the abstract state space is built dynamically, without having to a-priori generate the simulation space; and (b) The system to be verified is decomposed in a *network* of timed automata which communicate by synchronizing their actions. The global state space is generated directly from these automata, without having to a-priori compute the product automaton.

2 Preliminaries

2.1 Property-preserving abstractions

Abstractions. Let $\mathcal{S}$ be a set of *states*, and P be a set of *properties*. An *interpretation function* $\Pi : P \mapsto 2^{\mathcal{S}}$ associates with each property ϕ a set of states satisfying ϕ. Now, consider two sets of states $\mathcal{S}$ and $\mathcal{S}'$, referred to as the *concrete* and *abstract* state spaces, respectively. An *abstraction* from $\mathcal{S}$ to $\mathcal{S}'$ is a relation $\alpha \subseteq \mathcal{S} \times \mathcal{S}'$. Let $\Pi : P \mapsto 2^{\mathcal{S}}$ and $\Pi' : P \mapsto 2^{\mathcal{S}'}$ be concrete and abstract interpretation functions, respectively. We say that α is *safe* for P, with respect to Π, Π', iff for each property ϕ over P, for any concrete state $s \in \mathcal{S}$ such that $s \in \Pi(\phi)$, there exists an abstract state $s' \in \mathcal{S}'$ such that $s' \in \Pi'(\phi)$ and $(s, s') \in \alpha$, that is, s' is related to s by α. α is *exact* iff α and α^{-1} (the inverse relation) are safe.

Composition of abstractions. Given an abstraction α_1 from $\mathcal{S}$ to $\mathcal{S}'$ and an abstraction α_2 from $\mathcal{S}'$ to $\mathcal{S}''$, $\alpha_1 \circ \alpha_2$ is an abstraction from $\mathcal{S}$ to $\mathcal{S}''$, where $\circ$ denotes the composition of relations. The following facts can be derived directly from the definitions: (1) if both α_1 and α_2 are exact, then so is $\alpha_1 \circ \alpha_2$; (2) if one of α_1, α_2 is safe while the other one is either exact or safe, then $\alpha_1 \circ \alpha_2$ is safe.

2.2 Timed automata

Clocks, bounds and zones. Let $\mathcal{X} = \{x_1, ..., x_n\}$ be a set of variables called *clocks*, ranging over the positive reals $\mathbb{R}_{\geq 0}$. A *clock valuation* is a function $\nu : \mathcal{X} \mapsto \mathbb{R}_{\geq 0}$, assigning to each clock x a non-negative real value $\nu(x)$. For $X \subseteq \mathcal{X}$, $\nu[X := 0]$ is the valuation ν', such that $\forall x \in X.\ \nu'(x) = 0$ and $\forall x \notin X.\ \nu'(x) = \nu(x)$. For every $t \in \mathbb{R}_{\geq 0}$, $\nu + t$ is the valuation ν' such that $\forall x \in \mathcal{X}.\ \nu'(x) = \nu(x) + t$. A *bound* [11] over $\mathcal{X}$ is a constraint of the form of the form $x_i \# c$ or $x_i - x_j \# c$, where $1 \leq i \neq j \leq n$, $\# \in \{<, \leq, \geq, >\}$ and $c \in \mathbb{N} \cup \{\infty\}$. If we introduce a "dummy" clock variable x_0, taken to represent 0, then bounds can be uniformly written as $x_i - x_j \prec d$, where $0 \leq i \neq j \leq n$, $\prec \in \{<, \leq\}$ and $d \in \mathbb{Z} \cup \{\infty\}$. (For example, $x_1 > 3$ can be written as $x_0 - x_1 < -3$.) A bound $x_i - x_j \prec d$ is *stricter* than $x_i - x_j \prec' d'$ iff either $d < d'$ or $d = d'$ and $\prec = <, \prec' = \leq$. (We assume that $w < \infty$, for any real number w.) For instance, $x_i - x_j < 3$ is stricter than $x_i - x_j \leq 3$, which is stricter than $x_i - x_j < 4$, and so on. For two bounds b and b', $\min(b, b')$ (resp. $\max(b, b')$) is b (resp. b') if b is stricter than b', b' (resp. b) otherwise. A valuation ν satisfies a bound $x_i - x_j \prec d$ iff $\nu(x_i) - \nu(x_j) \prec d$, where, by convention, $\nu(x_0) = 0$.

A *zone* over $\mathcal{X}$ is a conjunction of bounds, $\bigwedge_{0 \leq i \neq j \leq n} x_i - x_j \prec_{ij} d_{ij}$, for $\prec_{ij} \in \{<, \leq\}$ and $d_{ij} \in \mathbb{Z}$. We denote by Z_{ij} the bound $x_i - x_j \prec_{ij} d_{ij}$. A valuation ν satisfies a zone Z iff ν satisfies Z_{ij}, for all $0 \leq i \neq j \leq n$. We often view a zone as the set of valuations satisfying it. Thus, we write $\nu \in Z$, to mean that ν satisfies Z, and $Z = \emptyset$, to mean that no valuation satisfies Z. We also write $Z \cap Z'$ to denote the zone Z'' corresponding to the intersection

of Z and Z', that is, such that $Z''_{ij} = \min(Z_{ij}, Z'_{ij})$, for all $0 \leq i \neq j \leq n$. Finally, we write $Z = Z'$ iff Z and Z' represent the same sets of valuations. Notice that $Z = Z'$ does not necessarily imply that Z and Z' are identical. For example, let $Z = 2 < x_1 < 3 \wedge 3 < x_2 < 4 \wedge 0 < x_2 - x_1 < 3$ and $Z' = 2 < x_1 < 3 \wedge 3 < x_2 < 4 \wedge 0 < x_2 - x_1 < 2$. Although the two zones are not syntactically identical, they are semantically equal (i.e., $Z = Z'$), since the bound $x_2 - x_1 < 3$ can be *strengthened* to the (stricter) bound $x_2 - x_1 < 2$. We say that a zone is in *canonical form* iff all its bounds are as strict as possible, that is, none can be strengthened to yield a semantically equal zone. In the sequel we assume that all zones considered are in canonical form. Let $\mathcal{Z}_{\mathcal{X}}$ denote the set of zones over $\mathcal{X}$.

Timed automata. A *timed automaton* (TA) [2, 14] is a tuple $A = (\mathcal{X}, Q, E, q_0, I)$, where: $\mathcal{X}$ is a finite set of clocks; Q is a finite set of *control locations*; E is a finite set of *edges* of the form $e = (q, Z, X, q')$, where $q, q' \in Q$ are the source and target locations, $Z \in \mathcal{Z}_{\mathcal{X}}$ is an enabling guard, and $X \subseteq \mathcal{X}$ is a set of clocks to be reset; q_0 is the initial control location; $I : Q \mapsto \mathcal{Z}_{\mathcal{X}}$ is a function associating with each control location q a *time-progress condition* $I(q)$ (we also write I_q). Figure 1(a) shows an example of a TA, with two clocks x and y, a single location with time progress condition *true* (i.e., $x \geq 0 \wedge y \geq 0$), and two edges a and b.

A *state* of a TA is a pair (q, ν), where $q \in Q$ is a location, and $\nu \in I_q$ is a valuation satisfying the time-progress condition of q. The semantics of A is the smallest set of such states, $\mathcal{S}_A$, such that:

1. $s_0 = (q_0, \mathbf{0}) \in \mathcal{S}_A$, $\mathbf{0}$ being the valuation assigning zero to all clocks;
2. if $(q, \nu) \in \mathcal{S}_A$ and there exists $e = (q, Z_e, X_e, q') \in E$ such that $\nu \in Z_e$, then $(q', \nu[X_e := 0]) \in \mathcal{S}_A$;
3. if $(q, \nu) \in \mathcal{S}_A$ and there exists $t \in \mathbb{R}_{\geq 0}$ such that $\nu + t \in I_q$, then $(q, \nu + t) \in \mathcal{S}_A$.

Due to the third rule above, $\mathcal{S}_A$ has generally a non-countable number of states.

3 Abstractions for TA

3.1 Simulation

This abstraction consists in mapping sets of concrete states to abstract (symbolic) states. It is based on the concept of *simulation graph*, a reachability graph used in KRONOS for checking safety properties [7] or, more recently, also liveness properties [4].

Consider a TA $A = (\mathcal{X}, Q, E, q_0, I)$, where $\mathcal{X} = \{x_1, ..., x_n\}$. Given a zone $Z \in \mathcal{Z}_{\mathcal{X}}$, a set of clocks $X \subseteq \mathcal{X}$, and an edge $e = (q, Z_e, X_e, q') \in E$, we define

the zone operators $\uparrow Z$, $Z[X := 0]$ and $\mathsf{e\text{-}succ}_e(Z)$, such that:

$$\uparrow Z = (\bigwedge_{i\in[0,n],j\in[1,n]} Z_{ij}) \wedge (\bigwedge_{i\in[1,n]} x_i - x_0 < \infty)$$

$$Z[X := 0] = (\bigwedge_{x_i,x_j\notin X} Z_{ij}) \wedge (\bigwedge_{(x_i\in X\vee x_j\in X)\wedge i,j\neq 0} x_i - x_j < \infty) \wedge (\bigwedge_{x_i\in X} 0 \leq x_i - x_0 \leq 0)$$

$$\mathsf{e\text{-}succ}_e(Z) = (Z \cap Z_e)[X_e := 0]$$

Notice that all operators above yield zones. Intuitively, $\uparrow Z \supseteq Z$ is the zone obtained from Z by eliminating all upper bounds of clocks (all bounds Z_{i0} are replaced by the trivial bound ∞), and represents the elapse of time. $Z[X := 0]$ is obtained after resetting all clocks in X to zero. Finally, $\mathsf{e\text{-}succ}_e(Z)$ corresponds to taking the part of Z satisfying the guard of e, then resetting some clocks as specified by e. Figure 2 shows examples of the application of the intersection, time-elapse, and reset operators.

A *symbolic state* S is a pair (q, Z), where $q \in Q$ is a location and $Z \in \mathcal{Z}_X$ is a zone. We write $(q', \nu) \in (q, Z)$ iff $q' = q$ and $\nu \in Z$. We write $(q, Z) = \emptyset$ iff $Z = \emptyset$. Let $e = (q, Z_e, X_e, q')$ be an edge. We define the post-condition operator post as follows:

$$\mathsf{post}_e((q, Z)) = (q', I_{q'} \cap \uparrow \mathsf{e\text{-}succ}_e(Z)) \tag{1}$$

That is, $\mathsf{post}_e(S)$ contains all states that can be reached from some state in S by, first taking a discrete transition by e, then letting some time pass in the new control location, while continuously satisfying its time-progress condition.

The *simulation space* of A is defined to be the smallest set of symbolic states $\mathcal{S}_A^{sim}$ such that:

1. $S_0^{sim} = (q_0, I_{q_0} \cap \uparrow\{\mathbf{0}\}) \in \mathcal{S}_A^{sim}$;
2. for any $S_1 \in \mathcal{S}_A^{sim}$ and any $e \in E$, if $S_2 = \mathsf{post}_e(S_1) \neq \emptyset$ then $S_2 \in \mathcal{S}_A^{sim}$.

In other words, $\mathcal{S}_A^{sim}$ is the set of all states obtained from S_0^{sim} by applying the post operator. $\mathcal{S}_A^{sim}$ can be infinite, as in the example of figure 1, where $\mathcal{S}_A^{sim}$ is displayed as a graph, the nodes of which are symbolic states, while the edges show the effect of applying post.

The *simulation abstraction* is defined to be the relation $\alpha_{sim} = \{(s, S) \in \mathcal{S}_A \times \mathcal{S}_A^{sim} \mid s \in S\}$.

Regarding the set of properties with respect to which reachability is defined, we consider the set $P = Q \times \mathcal{Z}_X$, that is, the set of all possible symbolic states. The interpretation function $\Pi : P \mapsto 2^{\mathcal{S}}$ is defined as: $\Pi(q, Z) = \{(q, \nu) \mid \nu \in Z\}$. Similarly, $\Pi^{sim} : P \mapsto 2^{\mathcal{S}_A^{sim}}$ is defined as: $\Pi^{sim}(q, Z) = \{(q, Z') \mid Z' \cap Z \neq \emptyset\}$.

Proposition 31 *The simulation abstraction α_{sim} is exact.*

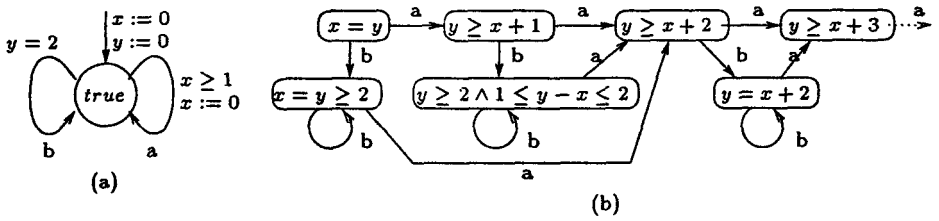

Fig. 1. A timed automaton (a), and its (infinite) simulation space (b).

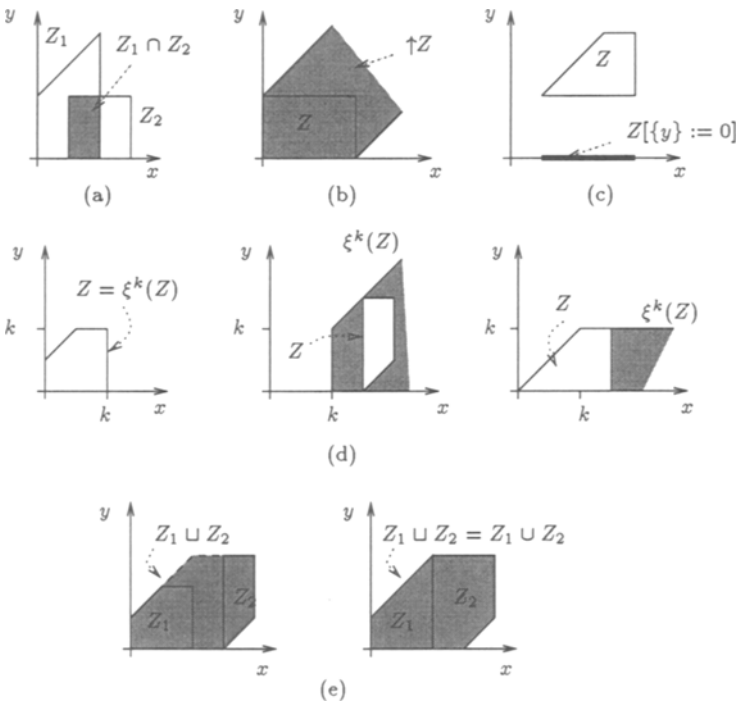

Fig. 2. Zone operations.

3.2 Extrapolation

The purpose of this abstraction is to ensure a finite number of symbolic states, since, as we have seen, the simulation space can be infinite. Extrapolation has been already used in KRONOS, as well as RTSPIN, where it is called *maximization* [20]. We explain informally the main idea, which is based on the (finite) region graph [2]. For any TA A, there is a maximal constant $k \in \mathbb{N}$ appearing in the zones of A (guards or time-progress conditions). In the example of figure 1, k equals 2. Now, consider a valuation ν and a clock x such that $\nu(x) > k$. It is easy to see that for any zone Z of A, the exact value of $\nu(x)$ is insignificant to whether ν belongs to Z or not, that is, for any other ν' which coincides with ν to all clocks but x, such that $\nu'(x) > k$, it holds $\nu \in Z$ iff $\nu' \in Z$. Consequently, once $\nu(x) > k$, $\nu(x)$ can be replaced by a "greater than k" value. We now formalize these notions.

Let $Z = \bigwedge_{0 \le i \ne j \le n} x_i - x_j \prec_{ij} d_{ij}$ be a zone over $\mathcal{X} = \{x_1, ..., x_n\}$. For each $k \in \mathbb{N}$, the *extrapolation function* $\xi^k : \mathcal{Z}_\mathcal{X} \mapsto \mathcal{Z}_\mathcal{X}$ is defined as follows (skip on first reading):

$$\xi^k(Z) = \bigwedge_{d_{ij} > k} x_i - x_j < \infty \quad \wedge \quad \bigwedge_{-d_{ij} > k} x_i - x_j < -k \quad \wedge \quad \bigwedge_{|d_{ij}| \le k} x_i - x_j \prec_{ij} d_{ij} \tag{2}$$

Figure 2(d) presents three examples of extrapolation, for $k = 2$, $\mathcal{X} = \{x, y\}$. (The white region is the zone before extrapolation. The filled region is the part added after extrapolation. Notice that the two rightmost extrapolated zones are unbounded.)

Intuitively, $\xi^k(Z)$ yields a zone $Z' \supseteq Z$, where:

- upper bounds greater than k are eliminated (first conjunct of equation 2);
- lower bounds greater than k are replaced by k (second conjunct of equation 2);
- all other bounds are preserved (third conjunct of equation 2).

Now, consider a TA A and let $k \in \mathbb{N}$ be a constant greater than or equal to the largest constant appearing in a zone of A. We write $\xi^k(q, Z)$ instead of $(q, \xi^k(Z))$. The *extrapolation space* of A with respect to k, denoted $\mathcal{S}^{xtr}_{A,k}$, is obtained by applying ξ^k to all states of the simulation space. Formally:

$$\mathcal{S}^{xtr}_{A,k} = \{\xi^k(S) \mid S \in \mathcal{S}^{sim}_A\}$$

Proposition 32 *For any TA A and any $k \in \mathbb{N}$, $\mathcal{S}^{xtr}_{A,k}$ is finite.*

Figure 3(a) shows the effect of applying the extrapolation abstraction for $k = 2$ to the example of figure 1. Notice that the part that changes is the infinite rightmost chain $y \ge x + 3, y \ge x + 4, ...$, which is replaced by a single symbolic state $y > x + 2$.

The *extrapolation abstraction* parameterized by k, is the relation $\alpha^k_{xtr} = \{(S, S') \in \mathcal{S}^{sim}_A \times \mathcal{S}^{xtr}_A \mid S' = \xi^k(S)\}$.

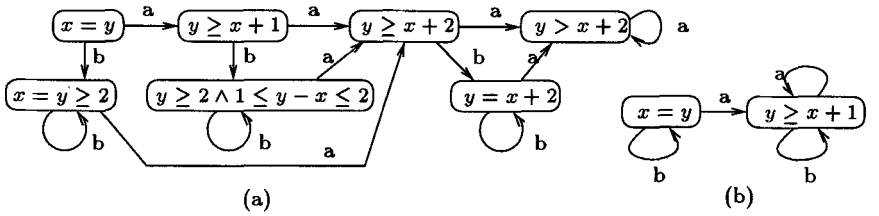

Fig. 3. Applying extrapolation (a) and (optimal) inclusion (b) to the example of figure 1.

The interpretation function Π^{xtr} is defined to be the same as Π^{sim}. Then, it is easy to prove that α^k_{xtr} is exact for any property ϕ such that the maximal constant appearing in ϕ is less than or equal to k. However, this is not generally true for properties involving constants greater than k, since it might be that such a property is reachable in the abstract space, while it is not in the concrete one. In any case, α^k_{xtr} is safe for any property. We now formally present these results.

For $k \in \mathbb{N}$, we define $\mathcal{Z}^k_\chi$ to be the set of all zones Z involving constants less than or equal to k. Let P_k be the set of properties $Q \times \mathcal{Z}^k_\chi$.

Proposition 33 *Let A be a TA and $k \in \mathbb{N}$ be a constant greater than or equal to the largest constant appearing in a zone of A. Then, α^k_{xtr} is exact with respect to P_k. Moreover, for all $m \in \mathbb{N}$, α^m_{xtr} is safe with respect to P.*

Remark 1. For the sake of simplicity, we have defined the extrapolation function with respect to a single constant k. In fact, it is straightforward to adapt the definitions for a set of constants c_{ij}, $0 \leq i \neq j \leq n$, one for each clock difference $x_i - x_j$. This permits to "optimize" the reduction, since a coarser abstraction is obtained. Preservation results are not affected.

3.3 Inclusion

Although finite, the number of states induced by extrapolation can still be large. This number can be reduced by using the *inclusion* abstraction, the main idea of which is the following. Consider two states S_1 and S_2 in the simulation space, such that S_1 is a subset of S_2. Then, for reachability properties, it is not necessary to examine neither S_1 (since any state belonging to S_1 belongs also to S_2), nor the successors of S_1, since each of them is a subset of the corresponding successor of S_2. Thus, states like S_1 can be eliminated. We formalize this in what follows.

Given a TA A, we say that a total function $\alpha_{inc} : \mathcal{S}^{sim}_A \mapsto \mathcal{S}^{sim}_A$ is an *inclusion abstraction* iff, for any $S \in \mathcal{S}^{sim}$, $\alpha_{inc}(S) \supseteq S$, where $(q, Z) \subseteq (q', Z')$ iff $q = q'$ and $Z \subseteq Z'$. (We regard, again, zones as sets of valuations, so that zone inclusion means set inclusion. $Z \subseteq Z'$ can be implemented by testing that each bound of Z is stricter than or equal to the corresponding bound of Z'.)

We define the *inclusion space* of A with respect to α_{inc} as the set of states $\mathcal{S}^{inc}_{A,\alpha_{inc}} = \alpha_{inc}[\mathcal{S}^{sim}_A]$, where $\alpha[\cdot]$ is the image of the relation α.

The above definitions allow for many possible inclusion abstractions. One inclusion abstraction for the simulation space of figure 1(b) is shown in figure 4(a), where the α_{inc} mapping is depicted by dashed arrowed lines. The corresponding inclusion space is shown in figure 4(b).

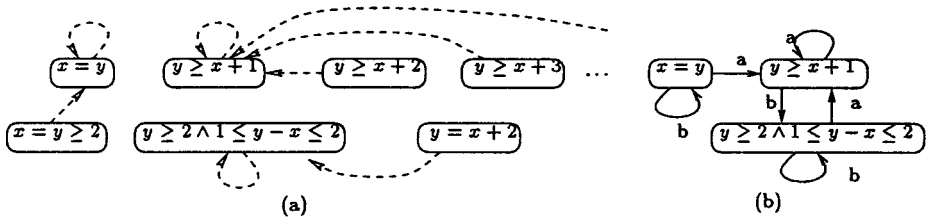

Fig. 4. Inclusion abstraction (a) and corresponding inclusion space (b).

An inclusion abstraction $\alpha^{\star}_{inc}$ is said to be *optimal* iff for any other α'_{inc}, $\alpha^{\star}_{inc}[\mathcal{S}^{sim}] \subseteq \alpha'_{inc}[\mathcal{S}^{sim}]$ (that is, $\alpha^{\star}_{inc}$ induces a smaller state space than α'_{inc}). In the case that the simulation space is finite, an optimal inclusion abstraction always exists. (In fact, there may be more than one optimal abstractions, but all of them induce the same inclusion space.) In the case that the simulation space is infinite, an optimal abstraction might still exist, as is the case for the TA of figure 1. The inclusion space induced by this optimal abstraction is shown in figure 3(b). On the other hand, consider a TA similar to the one of figure 1, where the guard $x \geq 1$ is replaced by $x = 1$. In this case, the simulation space as well as any inclusion space are infinite.

The interpretation function Π^{inc} is defined to be the same as Π^{sim}.

Proposition 34 *Any inclusion abstraction is exact.*

3.4 Activity

This abstraction permits to eliminate redundant clocks from a system. It has been introduced in [10] for the case of a single TA and is here generalized to a network of automata. The idea is that a clock should be considered *active* only when it *usefully* counts time, that is, from a point where the clock is reset, up to a point where the clock is tested. In any other case, the clock is inactive and can be ignored. We now formalize these notions.

Consider a TA $A = (\mathcal{X}, Q, E, q_0, I)$, where $\mathcal{X} = \{x_1, ..., x_n\}$. Given a control location $q \in Q$, $\mathsf{clk}(q) \subseteq \mathcal{X}$ is defined as the set of clocks x, such that either x appears in the time-progress condition I_q of q, or there exists an edge $e = (q, Z, X, q')$ in E, such that x appears in the guard Z of e.

Then, the function $\mathsf{act} : Q \mapsto 2^{\mathcal{X}}$, associating with each location q the set of active clocks in q, is defined as the least fix-point of the following system of equations (one equation for each location q):

$$\mathsf{act}(q) = \mathsf{clk}(q) \cup \bigcup_{(q,Z,X,q') \in E} \mathsf{act}(q') \setminus X \tag{3}$$

That is, x is active in q iff either x is in $\mathsf{clk}(q)$, or x is active in a location q' which can be reached from q by a sequence of edges, so that x is never reset along the sequence.

An algorithm to compute act is given in [10]. This algorithm works on the syntactic structure of the automaton, that is, its locations and edges, thus, it is extremely efficient.

Given a symbolic state $S = (q, Z)$, the *projection* of S to active clocks, denoted $S/_{\mathsf{act}}$, is a symbolic state (q, Z'), where Z' is the projection of Z to the set of active clocks of q. Formally:

$$(q, Z)/_{\mathsf{act}} = (q, \bigwedge_{x_i, x_j \in \mathsf{act}(q) \cup \{x_0\}} Z_{ij}) \tag{4}$$

The *activity space* $\mathcal{S}_A^{act}$ of a TA A is the set $\mathcal{S}_A^{act} = \{S/_{\mathsf{act}} \mid S \in \mathcal{S}_A^{sim}\}$. In other words, $\mathcal{S}_A^{act}$ is of *variable dimension*: for each $(q, Z) \in \mathcal{S}^{act}$, Z is a zone over $\mathsf{act}(q)$ (if $\mathsf{act}(q)$ is empty, the symbolic state reduces to just the control location q).

The *activity abstraction* is defined to be the relation $\alpha_{act} = \{(S, S') \in \mathcal{S}_A^{sim} \times \mathcal{S}_A^{act} \mid S' = S/_{\mathsf{act}}\}$.

The interpretation function Π^{act} is defined to be the same as Π^{sim}.

Proposition 35 *The activity abstraction α_{act} is exact for any property $\phi = (q, Z)$, such that the set of clocks appearing in Z is a subset of* $\mathsf{act}(q)$.

Remark 2. The above proposition claims that α_{act} is exact for ϕ if the latter refers to clocks which are active in q. In fact, it is easy to extend the above definitions so that the activity abstraction is exact for any property (q, Z). Indeed, it suffices to add all clocks appearing in Z in the set of clocks initially active in q, $\mathsf{clk}(q)$, and compute the fix-point equations defined in 3 accordingly.

3.5 Convex hull

This abstraction provides a considerable reduction of the state space, permitting to keep a single zone Z with each control location q of the system. In general, there will be many zones $Z_1, ..., Z_n$ associated with a location q, in any of the abstract spaces defined previously. However, $(q, \bigcup_{i=1,...,n} Z_i)$ is not a symbolic state, since the union of two zones is generally a *non-convex* set, that is, cannot be represented as a conjunction of constraints (i.e., a zone). The *convex hull* of two zones is by definition convex, therefore, one can abstract $\bigcup_{i=1,...,n} Z_i$ by $\bigsqcup_{i=1,...,n} Z_i$, where $\sqcup$ denotes the convex hull. On the other hand, convex hulls

are generally supersets of unions, meaning that states which are not reachable in the concrete space might be so in the abstract one. This is why this abstraction is not exact, although it is safe. Convex hull abstractions have been used also by [12, 21, 3]. The definitions are given in what follows.

Given two zones Z and Z', the *convex hull* of Z and Z', denoted $Z \sqcup Z'$, is defined as the smallest (with respect to set inclusion) zone Z'' such that $Z \subseteq Z''$ and $Z' \subseteq Z''$. ($Z \sqcup Z'$ can be obtained operationally as follows: $(Z \sqcup Z')_{ij} = \max(Z_{ij}, Z'_{ij})$.) Two examples are shown in figure 2(e).

The *convex-hull space* of a TA A is the set $\mathcal{S}_A^{ch} = \{(q, Z) \mid Z = \bigsqcup\{Z' \mid (q, Z') \in \mathcal{S}_A^{sim}\}\}$. [2]

The *convex-hull abstraction* is defined to be the relation:

$$\alpha_{ch} = \{((q, Z), (q, Z')) \in \mathcal{S}_A^{sim} \times \mathcal{S}_A^{ch} \mid Z \subseteq Z'\}.$$

The interpretation function Π^{ch} is defined to be the same as Π^{sim}.

Proposition 36 *The convex-hull abstraction α_{ch} is safe.*

4 Model checking using abstractions

The reachability analysis is implemented in KRONOS as a *breadth-first* (BFS) or *depth-first* (DFS) generation of the abstract state space, starting from an initial state, and checking whether a final state $(\hat{q}, \hat{Z})$ is reachable from an initial state (q_0, Z_0). [3]

Figure 5 shows the DFS procedure. $\mathcal{S}^\alpha$ is the set of visited (abstract) states, initialized to $\{(q_0, Z_0)\}$. For each newly-generated state (q, Z), it is checked whether the latter satisfies the property $(\hat{q}, \hat{Z})$, and, if so, a sample trail is returned as output (in this case, by simply running through the DFS stack). Otherwise, (q, Z) is stored to the set of visited states $\mathcal{S}^\alpha$, and the search goes on to explore all successor states which have not been visited yet.

The search is parameterized by an abstraction α, which is either α_{sim}, or $\alpha_{sim} \circ \alpha'$, α' being itself a composition of some of the other four abstractions, $\alpha_{xtr}, \alpha_{inc}, \alpha_{act}, \alpha_{ch}$. The correctness of the method comes from propositions 31, 33, 34, 35, 36, and the fact that composition of abstractions respects property preservation (see section 2.1). Depending on α, the functions post_e, **store** and **visited** are modified appropriately, to implement the chosen abstractions. We explain this in what follows.

Extrapolation and Activity. These abstractions are implemented by modifying the successor function post_e. When none of these abstractions is used, then

[2] We should note that, in this definition, $\bigsqcup\{Z' \mid (q, Z') \in \mathcal{S}_A^{sim}\}$ is actually the *smallest upper bound* of the set $\{Z' \mid (q, Z') \in \mathcal{S}_A^{sim}\}$, in the lattice of zones with respect to set inclusion. This is because the set $\{Z' \mid (q, Z') \in \mathcal{S}_A^{sim}\}$ may be infinite.

[3] Many interesting properties can be formulated in this way. In particular, this is true for *invariants*, informally stated as "in all reachable states p holds", and *real-time bounded-response* properties informally stated as "p will hold in at most t units of time". We refer the reader to [7] for more details.

```
ReachDFS^α ((q, Z), (q̂, Ẑ)) :
  if (q = q̂ and Z ∩ Ẑ ≠ ∅) then
    return((q̂, Ẑ) is reachable) ;
    output(DFS stack) ;
  else
    store^α((q, Z), S^α) ;
    for-each (e ∈ E) do
      (q', Z') := post_e^α(q, Z) ;
      if (Z' ≠ ∅ and not visited^α(q', Z')) then
        ReachDFS^α((q', Z'), (q̂, Ẑ)) ;
```

Fig. 5. Depth-first reachability using abstractions.

post_e^α is simply post_e, defined in equation 1. When extrapolation is used, then $\mathsf{post}_e^\alpha(q, Z)$ is $\xi^k(\mathsf{post}_e(q, Z))$, where k is found as explained in section 3.2. When activity is used, then $\mathsf{post}_e^\alpha(q, Z)$ is $(\mathsf{post}_e(q, Z))/_{\mathsf{act}}$. When both abstractions are used, both operators are applied (the order does not matter).

Inclusion. This abstraction is implemented by modifying the test $\mathsf{visited}^\alpha(q', Z')$. When inclusion is not used, this test is $(q', Z') \in \mathcal{S}^\alpha$. Otherwise, the test becomes $\exists (q', Z'') \in \mathcal{S}^\alpha.\ Z' \subseteq Z''$.

Convex hull. This abstraction is implemented by, first, changing visited as in the case of inclusion, and second, modifying also the storing procedure as follows. When convex hull is used, $\mathsf{store}^\alpha((q, Z), \mathcal{S}^\alpha)$ has the following effect: either there is already a state (q, Z'') in $\mathcal{S}^\alpha$, in which case it is replaced by $(q, Z'' \sqcup Z)$; or there is no such state, thus, (q, Z) is added to $\mathcal{S}^\alpha$. When convex hull is not used, (q, Z) is simply added to $\mathcal{S}^\alpha$.

A final comment needs to be made on the incompleteness of the convex-hull abstraction: if a state is reachable in the abstract graph then no conclusion can be made about the concrete graph. A (partial) solution to this is to try to follow the diagnostic trail given by the DFS (i.e., the sequence of transitions fired) using the exact successor function post. This means generating the concrete counterpart of the abstract trail. If the entire transition sequence can be generated then the reachability property indeed holds, otherwise the approximation is too coarse. Notice that this method cannot be generalized, for instance, by simply testing all counter-example trails found in the abstract graph, since their concrete counterparts constitute only a part of the set of trails in the concrete graph.

5 Examples

5.1 The FDDI communication protocol

FDDI (Fiber Distributed Data Interface) [15] is a high-performance fiber-optic token-ring LAN (local-area network). An FDDI network is composed by N iden-

tical stations and a ring, where the stations can communicate by *synchronous* messages with high priority or *asynchronous* messages with low priority.

We are interested here in the verification of one aspect of the temporal mechanism of the protocol, namely, the *bounded time for accessing the ring* (BTAR) property, stated as: "the time elapsed between two consecutive receptions of the token by a given station is bounded by c_N". (c_N is a constant depending on the number of stations N).

The system has already been specified and verified with KRONOS, in [8], using a backward fix-point computation, and in [7], using a forward reachability analysis (i.e., exploring the simulation space). In the first case, 8 stations has been the maximum the tool could handle, while in the second, the limit has been 12 stations.

Here we show how enhancing the reachability analysis with abstractions leads to a more efficient verification of this system, both in time and space, thus permitting to verify the property on a system of up to 50 stations.

Specification. The system for N stations is modeled as a TA $\mathbf{FDDI}_N$ which is obtained as the parallel product of the model of the ring for N stations $\mathbf{Ring}_N$, and the model of each station, $\mathbf{Station}_i$, $i = 1, ..., N$. It is worth noting that the size of $\mathbf{FDDI}_N$ (in locations, transitions and clocks) is linear on the number of stations N. Since the computation of the model of the global system is not costly, on-the-fly verification is not needed in this case. Instead, we apply the reachability analysis to the global model, using the breadth-first-search technique.

Results. Figure 6 shows, in logarithmic scale, the performance results [4]. We can conclude from figure 6(a) that exact abstractions generate an exponential number of symbolic states [5], while their combination with the convex hull abstraction reduces the cost from exponential to polynomial on the number of stations. However, it is important to notice that activity reduces the number of symbolic states generated by half when the convex hull abstraction is not used.

From figure 6(b), we can make the same conclusions about the complexity on the number of stations as those for the size of the state-space generated. However, it turns out that the benefits from the activity abstraction are much more important in terms of time reduction than in terms of state-space size reduction, even when combined with the convex-hull abstraction. The reason for this is that reducing by activity the number of clocks of the system, leads to a more compact representation of zones (i.e. a gain in memory), and to a more efficient computation of the operations on zones. In this case, activity reduces the number of clocks from $2N + 1$, for the original model (2 clocks for each station,

[4] The model was verified using the simulation abstraction (denoted "sim" in the figure) alone, or combined with the convex-hull ("ch") or the activity ("act") abstractions. Extrapolation and inclusion have no effect in this case because of the structure of the model.

[5] In this example, the exponential complexity is induced by the temporal aspect of the system, since the size of its control is linear.

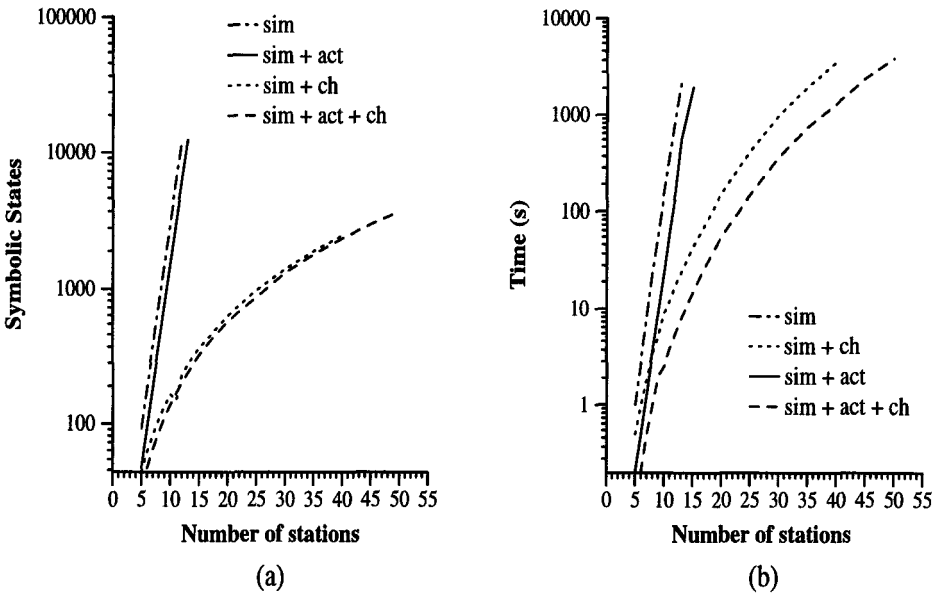

Fig. 6. Results of verification of FDDI protocol using abstractions: state-space size (a) and time (b).

plus a clock for the ring), to $N + 1$, because the clock of the ring and one of the clocks of each station are never active simultaneously. Hence the important gain, both in memory and time, even if the number of symbolic states computed is nearly the same.

5.2 Fischer's mutual-exclusion protocol

This is a real-time mutual-exclusion protocol which has become a benchmark example, thus we ommit its description here (see [9] for more details).

Results. Diagrams (a) and (b) in figure 7 display the results of using abstractions, in linear and logarithmic scale, respectively. [6] Some conclusions coming from these results are the following.

[6] "x" stands for extrapolation, "inc" for inclusion. "act" for activity, and "ch" for convex hull. The "+" symbol stands for combination of abstractions. Not all combinations are shown: first, the simulation graph is infinite; second, by definition, convex hull is more general than inclusion; "x+inc+act" turns out to yield exactly the same results as "x+act"; finally, "x+ch" yields almost the same results as "act+ch". Concerning time and memory costs, the largest case treated (9 processes, about 600,000 symbolic states generated) has consumed 2 hours of CPU time and 180 megabytes, on a Sparc-station 20 with 224 megabytes of memory. We should mention that computing the abstractions does not result in a significant time consumption, that is, the overhead is a matter of seconds.

1. Combination of abstractions is definitely useful in absolute terms, e.g., compare the performances of α_{inc} and $\alpha_{xtr} \circ \alpha_{inc}$ for 5 processes, or α_{xtr} and $\alpha_{xtr} \circ \alpha_{inc} \circ \alpha_{act}$ for 6 processes.
2. The convex-hull abstraction radically reduces the state space, permitting to handle up to 9 processes.
3. The complexity of the problem remains exponential, even with the use of convex hull.

Comparison. Fischer's protocol has been previously treated by KRONOS using just extrapolation, and without on-the-fly generation of the state space (i.e., the syntactic product of the TA had to be constructed a priori). This approach was able to handle up to 5 processes, consuming about 140 seconds of CPU time. The limit reported in [17] using the tool UPPAAL has also been 5 processes, which consumed 600 seconds of CPU time. [7] Similar results have been reported in [18] for the tool HYTECH [13].

In [3,5], BDDs (binary decision diagrams) have been used to verify the protocol for up to 10 and 14 processes, respectively. [3] uses a safe abstraction that corresponds to the convex hull, while [5] uses an exact discretization of the state space. The main drawback of these BDD-based methods is that they are quite sensitive to the size of constants: in both case studies above, the values of δ, Δ were assumed to be 1 and 2, respectively. Also, in case the property fails to hold, the BDD encoding the set of reachable states does not contain enough information in order to provide a counter-example, so that some kind of enumerative exploration needs to be also available. (In [5], counter-examples are generated by re-starting the exploration using a BFS, once the BDD is found to intersect the property in question).

Finally, the protocol has been also treated in [18], for up to 7 processes, using a *formula-quotienting* construction, which is not an explorative model-checking approach, thus, cannot be directly compared to ours. [8]

6 Conclusions and perspectives

We proposed abstractions as an efficient way to perform verification of reachability properties. In summary, the simulation abstraction is the formalization of the coarse reachable state space upon which the algorithm is based; extrapolation is in some cases inevitable so that the algorithm terminates; the inclusion, convex-hull and activity abstractions are equally important in practice, since

[7] This was on a different machine than ours, and possibly with different constants δ, Δ. We have tried UPPAAL version 2.02, March'97, (command `verifyta -S`, which enables all state-space reductions) on our machine, for 5 and 7 processes, using the same constants. UPPAAL consumed 485 seconds of CPU time for 5 processes, and hadn't finished for 7 processes after more than 14 hours of CPU time, having consumed 70MB of memory.

[8] In [16], the same approach has been used to verify an acyclic version of the protocol for up to 50 processes.

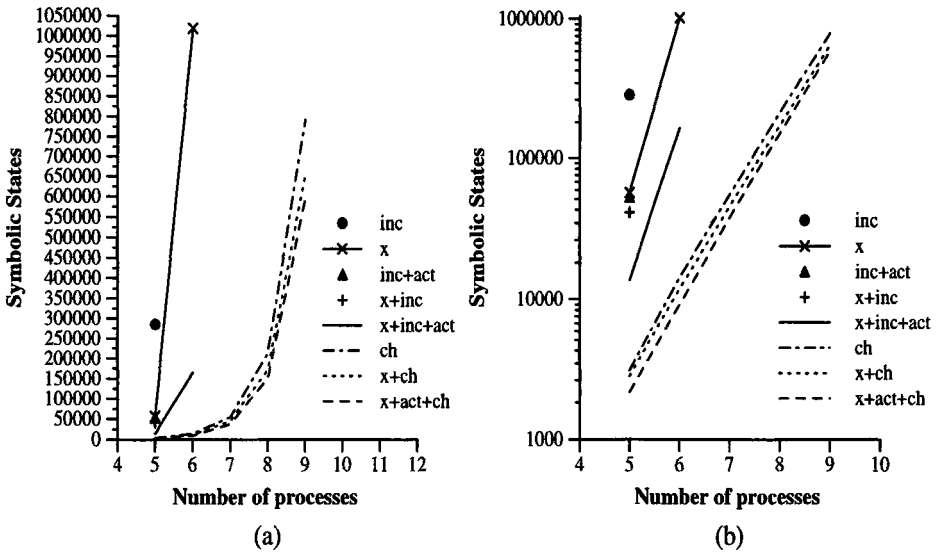

Fig. 7. Results of verification of Fischer's protocol using abstractions.

they reduce the size of the state space: inclusion and convex-hull reduce the *number* of symbolic states generated, whereas as activity also reduces the *size* needed to store each symbolic state.

Apart from convex hull, all other abstractions are exact. The former is still useful, however: If a state is not reachable in the abstract space, it is certainly not reachable in the concrete space either. If a state is reachable in the abstract space, one can always examine the path leading to this state, by re-executing the transitions without applying the abstraction. If the state is indeed reachable then a diagnostic trail is found. Otherwise, the search can continue in the same manner, providing some confidence in the system verified, without, however being definitely conclusive.

Experimental results allow us to infer that our abstractions deal quite well with two factors of exponential growth of the state space, namely, the number of clocks, and the size of constants used in the model. Regarding the third factor of exponential growth, that is, the number of components constituting the system, although these abstractions provide an improvement of performance in absolute terms, they cannot avoid the exponential complexity in general. Notice, however, that this is achieved in the FDDI example, with the help of the convex-hull abstraction.

Regarding perspectives, in the short term, we plan to complete the prototype implementation, namely, by programming variable-dimension structures, and optimal storage techniques. In the long term, compositional methods like the ones in [1, 18], need to be studied more thoroughly.

References

1. P. Abdulla and B. Jonsson. Verifying networks of timed processes. 1997. To appear.
2. R. Alur. *Techniques for automatic verification of real-time systems.* PhD thesis, Stanford University, 1991.
3. F. Balarin. Approximate reachability analysis of timed automata. In *Proc. 17th IEEE Real-Time Systems Symposium*, 1996.
4. A. Bouajjani, S. Tripakis, and S. Yovine. On-the-fly symbolic model checking for real-time systems. In *Proc. of the 18th IEEE Real-Time Systems Symposium*, 1997.
5. M. Bozga, O. Maler, A. Pnueli, and S. Yovine. Some progress in the symbolic verification of timed automata. In *Proc. of the 8th Conference on Computer-Aided Verification*, 1997.
6. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages*, 1977.
7. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Hybrid Systems III.* LNCS 1066, 1996.
8. C. Daws, A. Olivero, and S. Yovine. Verificación automática de sistemas temporizados utilizando KRONOS. In *Proc. Jornadas de Informática y Telecomunicaciones de la IEEE (sección Uruguay)*, 1996.
9. C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions (full version). Technical Report 97-08, Verimag, october 1997. http://www.imag.fr/VERIMAG/PEOPLE/Conrado.Daws.
10. C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proc. 17th IEEE Real-Time Systems Symposium, RTSS'96*, 1996.
11. D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. 1st Intl. Workshop on Computer-Aided Verification*, 1989.
12. N. Halbwachs. Delay analysis in synchronous programs. In *5th Conference on Computer-Aided Verification.* LNCS 697, 1993.
13. T. Henzinger, P. Ho, and H. Wong-Toi. Hytech: The next generation. In *Proc. 16th IEEE Real-time Systems Symposium*, 1995.
14. T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 1994.
15. R. Jain. *FDDI handbook: high-speed networking using fiber and other media.* Addison-Wesley, 1994.
16. Kristoffersen, F. Laroussinie, K. Larsen, P. Petterson, and W. Yi. A compositional proof of a real time mutual exclusion protocol. In *Proc. of the 7th Intl. Conf. on the Theory and Practice of Software Development*, 1997.
17. K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: Compact data structures and state-space reduction. In *Proc. of the 18th IEEE Real-Time Systems Symposium*, 1997.
18. K. Larsen, P. Petterson, and W. Yi. Compositional and symbolic model-checking of real-time systems. In *Proc. 16th IEEE Real-Time Systems Symposium*, 1995.
19. C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 1995.
20. S. Tripakis and C. Courcoubetis. Extending promela and spin for real time. In *TACAS'96.* LNCS 1055, 1996.
21. H. Wong-Toi. *Symbolic Approximations for Verifying Real-Time Systems.* PhD thesis, Stanford University, 1995.