VERIFICATION OF DENSE TIME PROPERTIES USING THEORIES OF UNTIMED PROCESS ALGEBRA

Matti Luukkainen
University of Helsinki, Department of Computer Science
PO Box 26, FIN-00014 University of Helsinki, FINLAND
mluukkai@cs.helsinki.fi

Abstract The article shows how untimed process algebraic methods can be used in verifying a category of dense time properties from a restricted class of timed automata. The work is based on the known results that if we restrict ourselves to a certain class of timed automata, then most of the interesting dense time properties are verifiable using discrete time methods. Those results are refined here in order to fit the process algebraic context, and furthermore, reduction strategies based on behavior equivalences, compositionality and abstraction are indicated. The usability of the method is evaluated with two case studies.

Keywords: Formal verification, Timed automata, Process algebra, Compositionality

1. INTRODUCTION

Because of the state explosion problem, a useful principle in algorithmic verification is that *the information unnecessary for a verification task should be abstracted away.* Especially, we would like to see in the context of systems specified with timed automata when it is sufficient to use discrete time algorithms but still say something about the system behavior in dense time.

In Henzinger et al., 1992 it was shown that for systems specified with timed transition systems, a large class of dense time properties can be verified by using discrete time algorithms. These results were generalized to timed automata in Bošnački, 1999. In this article we show how the methods used in standard untimed process algebra, and the related tools, can be used in verifying this category of dense time properties

from the timed automata descriptions. We have refined the previously known results in order to fit the process algebraic context, and furthermore, some efficient reduction strategies based on behavior equivalences, abstraction and compositionality are advocated.

2. TIMED AUTOMATA

Informally, a timed automaton (Alur and Dill, 1994) is just a transition system extended with a set of a special type of variables, namely *clocks* which are used to control the executability and urgency of transitions with respect to real time.

Let X be a finite set of non-negative valued variables, namely clocks. Let Φ be a set of predicates clocks, defined as the smallest set satisfying: $\phi ::= true \mid x \sim c \mid \phi \wedge \phi$, where $x \in X$, $c \in \mathrm{N}$ and $\sim \in \{<, \leq, =, \geq, >\}$.

A timed automaton is a 6-tuple (S, A, X, R, I, s_0), where

- S is a finite set of control states,
- A is a finite set of transition labels or actions,
- $X \subseteq X$ is a finite set of real-valued clocks,
- $R \subseteq S \times A \times \Phi \times 2^X \times S$ is the transition relation,
- $I : S \rightarrow \Phi$ is a function that labels control states with time progress conditions,
- $s_0 \in S$ is the initial control state.

In a transition $(s, a, \phi, Y, s') \in R$, s is the source and s' the destination control state, $\phi \in \Phi$ is the guard of the transition. and $Y \subseteq X$ denotes the set of clocks that are reset to zero during the transition.

A valuation $v \in V$ is a function that assigns a non-negative real value to each of the clocks, i.e. for every $x \in X : v(x) \in \mathrm{R}^{\geq 0}$. If X denotes a set of clocks, valuation $v[X := 0]$ is defined as follows: if $x \in X$ then $v[X := 0](x) = 0$, else $v[X := 0](x) = v(x)$. If $t \in \mathrm{R}^{\geq 0}$, and v is a valuation, then $v + t$ is a valuation such that $(v + t)(x) = v(x) + t$ for all the clocks x.

A *state* of a timed automaton is its control state and a valuation of the clocks. Let $Q \in S \times V$ be the set of states of a timed automaton. A timed automaton can evolve in two ways, either executing a discrete transition or letting time pass without changing the control state. Possible transitions are formulated by the two rules below.

- Let (s, a, ϕ, Y, s') be a transition. The state (s, v) has a *discrete transition* labeled with a to (s', v'), if $\phi(v)$ evaluates to true and $v' = v[Y := 0]$.
- Let $t \in \mathrm{R}^{\geq 0}$. The state (s, v) has a time transition labeled with t to $(s, v + t)$ if for all $t' \leq t$ the time progress condition $I(s)(v + t')$ is true.

As can be noticed, the *state space* of a timed automaton can be defined with a transition system. Its alphabet consists of the transition

labels of the timed automaton (the discrete transitions) and positive real values (the time transitions). Formally, given a timed automaton $TA = (S, A, X, R, I, s_0)$, its state space is defined by a transition system $S_{TA} = (S \times \mathcal{V}, A \cup \mathbb{R}^{\geq 0}, R^t, (s_0, v_0))$, where R^t is the smallest set containing all discrete and timed transitions of the timed automaton (which are given by the two rules above), and v_0 is a valuation that assigns the value zero to all the clocks.

The most natural way of modeling a system is to describe each of the components with a separate timed automaton. The behavior of the entire system is then defined by the timed automaton resulting from the parallel composition (denoted with $||$) of the component automata.

Now we will give an automata theoretic semantics of timed automata. A *timed word* is a pair $(\bar{\sigma}, \bar{t})$, where $\bar{\sigma} = \sigma_1, \sigma_2 \ldots$ is an infinite sequence of transition labels $\sigma_i \in A$, and $\bar{t} = t_1, t_2 \ldots$ is an infinite, monotone and progressive sequence of time values. The intuition behind the timed word is that a transition labeled with σ_i has occurred at time t_i from the beginning of the observation. All the possible timed words with time domain $\mathbb{T}$ are denoted $TW_{\mathbb{T}}$.

If we add to the definition of timed automaton *a set of accepting states* $F \subseteq S$, then it can be thought of as a *timed Büchi-automaton* which accepts a set of timed words.

An *execution* of timed automaton *TA* which *corresponds* to the timed word $(\bar{\sigma}, \bar{t})$ is an infinite sequence

$$\bar{q} = q_1 \xrightarrow{t_1} q_1' \xrightarrow{\sigma_1} q_2 \xrightarrow{t_2 - t_1} q_2' \xrightarrow{\sigma_2} q_3 \xrightarrow{t_3 - t_2} q_3' \xrightarrow{\sigma_3} q_4 \ldots,$$

in the S_{TA}, so each q_i, q_i' are of the form (s, v). The intuition behind the execution is that at the ith step, the automaton first lets time progress for $t_i - t_{i-1}$ units and then executes a transition labeled with σ_i.

A timed automaton *TA accepts* the timed word $(\bar{\sigma}, \bar{t})$ if it has an execution $\bar{q}$ corresponding to that word, starting from the initial state (s_0, v_0) and which contains infinite occurrences of states (s_i, v_i) such that $s_i \in F$. Language accepted by a timed automaton *TA* is defined as usual: $[\![TA]\!] = \{(\bar{\sigma}, \bar{t}) \mid TA$ has an accepting computation corresponding to the word $(\bar{\sigma}, \bar{t})\}$.

In the automata theoretic setting, the question whether a system S satisfies a requirement φ (denoted $S \models \varphi$) can be formulated as follows. The system and the property are defined as two timed Büchi-automata TA_S and TA_φ. The automata define two sets of timed words $[\![TA_S]\!]$ and $[\![TA_\varphi]\!]$. Now the question whether $S \models \varphi$ is equivalent to the inclusion $[\![TA_S]\!] \subseteq [\![TA_\varphi]\!]$. The inclusion can be replaced by the emptiness check of $[\![TA_S]\!] \cap [\![TA_{\overline{\varphi}}]\!]$, where $TA_{\overline{\varphi}}$ is a timed Büchi-automata that defines the negation of the property φ.

Intersection $[\![TA_1]\!] \cap [\![TA_2]\!]$ of the languages accepted by two timed Büchi-automata equals the language accepted by the product automaton $[\![TA_1 \times TA_2]\!]$ (Alur and Dill, 1994). In the special case where all the states of the other automata, e.g. TA_1 are accepting, the product automaton $TA_1 \times TA_2$ corresponds to the parallel composition $TA_1 \,||\, TA_2$, where the accepting states are those in which the component corresponding to TA_2 is in accepting state. Since the automaton TA_S that describes the system usually has all the states as accepting ones, the verification can be done by checking whether $[\![TA_S \,||\, TA_{\overline{\varphi}}]\!] = \emptyset$.

3. VERIFICATION

It is known that several interesting verification problems are decidable for timed automata. Essentially the method of showing the decidability, and also the way of conducting the verification is the construction of a *region graph,* which is a finite abstraction of the state space of a timed automaton.

The dense time verification methods based on finite abstractions are computationally quite hard. Furthermore, the decidability of the dense time methods does not allow the clock values to be compared against arbitrary real values, but instead just integers. A question that now arises is, whether we could use discrete time verification algorithms in deducing dense time properties.

In Henzinger et al., 1992 it was shown that for systems specified with timed transition systems, a large class of dense time properties can be verified by using discrete time algorithms. These results were generalized to timed automata in Bošnački, 1999.

We have taken these known results as our starting point in developing our verification methodology. Since the theory in Henzinger et al., 1992 and Bošnački, 1999 is quite involved, we just rephrase here the most interesting consequences from our point of view. See the full version of this paper (M. Luukkainen, 2001) for full treatment and motivation for the result.

Let Π be a set of timed words. Let us denote with $\mathbb{Z}(\Pi)$ a subset of timed words of Π, which consists only of those words where the observation times t_i are natural numbers, formally $\mathbb{Z}(\Pi) = \Pi \cap TW_{\mathbb{N}}$. In other words, if $[\![TA]\!]$ characterizes the behavior of an automaton *TA* within dense time semantics, $\mathbb{Z}(TA)$ would be its *interpretation in the discrete time model.* Now the question that we are interested in is, what can be concluded about the dense time behavior of an automaton *TA* by verifying $\mathbb{Z}(TA) \subseteq \mathbb{Z}(TA_\varphi)$, or $\mathbb{Z}(TA) \cap \mathbb{Z}(TA_{\overline{\varphi}}) = \emptyset$, for a property φ. Thus,

we want to know what dense time properties of systems can be deduced by using discrete time methods.

Firstly, let us call a timed automaton where relations only of the form $\leq$, $=$, $\geq$ are used as a *weakly constrained* timed automaton. Analogously let us call a timed automaton where only $<$, $>$ or the trivial constraint *true* are used as a *strongly constrained* timed automaton.

Now the following theorem (Henzinger et al., 1992; Bošnački, 1999) describes the relation between discrete and dense time verification.

Theorem 1 *Let TA_1 and TA_2 be two weakly constrained and TA_3 a strongly constrained timed Büchi automata.*

1. $[\![TA_1]\!] \subseteq [\![TA_3]\!]$ *if and only if* $\mathbb{Z}(TA_1) \subseteq \mathbb{Z}(TA_3)$.
2. $[\![TA_1]\!] \cap [\![TA_2]\!] = \emptyset$ *if and only if* $\mathbb{Z}(TA_1) \cap \mathbb{Z}(TA_2) = \emptyset$.
3. $\mathbb{Z}(TA_1) \cap \mathbb{Z}(TA_2) = \mathbb{Z}(TA_1 \parallel TA_2)$.

The above theorem guarantees the following: if a system is described with weakly constrained timed automaton TA_S and the negation of the correctness requirements is given as another weakly constrained automaton, say $\overline{\varphi}$, the verification task whether $S \models \varphi$ can now be reduced to checking whether the language accepted by the parallel composition, of TA_S and $TA_{\overline{\varphi}}$ is empty in the discrete time domain. Since the result of the parallel composition itself is just one timed automaton (with a set of accepting states), we can now concentrate on checking the language emptiness of a single timed automaton in the discrete time interpretation.

As stated in the previous section, a timed automaton TA accepts a timed word $(\overline{\sigma}, \overline{t})$ if and only if the corresponding state space S_{TA} has a path which corresponds to the timed word, and the path visits an accepting state infinitely often. So, in order to check the emptiness of TA one could observe whether the condition holds in the transition system S_{TA}. But since S_{TA} is inherently an infinite structure, even in the discrete time interpretation that we now only consider, it is not practical for verification purposes.

In the discrete time domain, the reason for infiniteness is the fact that clock values can potentially increase without bounds. In every timed automaton there exists the greatest value x_c against which a clock $x \in X$ is compared. When the value of x is greater than x_c, the further increase is no more meaningful to the behavior of the system. So, keeping this in mind, we can modify the rules for transitions of timed automata in the following way (the first rule is unaltered):

- Let (s, a, ϕ, Y, s') be a transition. The state (s, v) has a *discrete transition* labeled with a to (s',v'), if $\phi(v)$ evaluates to true and $v' = v[Y := 0]$.
- Let $t \in \mathbb{R}^{\geq 0}$. The state (s, v) has a *time transition* labeled with t to (s, v'),

where $v'(x) = v(x) + t$ if $x \leq x_c$, else $v'(x) = x_c$,
if for all $t' \leq t$ the time progress condition $I(s)(v + t')$ is true.

Now we can define the *reduced state space* RS_{TA}, a finite structure that can be used to derive the information about the executions of the timed automaton TA in the discrete time interpretation.

Definition 2 *Given a timed automaton* $TA = (S, A, X, R, I, s_0)$, *its reduced state space is defined by a transition system* $RS_{TA} = (S \times \mathcal{V}, A \cup \mathbb{R}^{\geq 0}, R^t, (s_0, v_0))$, where R^t *is the smallest set containing all discrete and timed transitions (given by the two rules above) of the timed automaton in the discrete time domain.*

The next theorem summarizes two important properties of transition systems corresponding to reduced state spaces. The proofs can be found in the full version of the paper (M. Luukkainen, 2001).

Theorem 3 *Let TA and TA′ be two timed automata.*

1 *There is a path* $\bar{q}_S = q_1 \xrightarrow{t_1} q_1' \xrightarrow{\sigma_1} q_2 \xrightarrow{t_2} q_2' \xrightarrow{\sigma_2} \ldots$ *in* S_{TA} *if and only if there is a corresponding path* $\bar{q}_{RS} = q_1 \xrightarrow{1} q_{11} \xrightarrow{1} \ldots q_{1t_1-1} \xrightarrow{1} q_1' \xrightarrow{\sigma_1} q_2 \xrightarrow{1} q_{21} \xrightarrow{1} \ldots q_{2t_2-1} \xrightarrow{1} q_2' \xrightarrow{\sigma_2} \ldots$ *in* RS_{TA}.

2 $RS_{TA} \parallel RS_{TA'}$ *is strongly bisimilar to* $RS_{TA \parallel TA'}$.

The first part gives the needed evidence that the use of RS_{TA} is actually enough to determine whether the language accepted by TA is empty. The second part indicates that the transitions corresponding to time progress behave just like ordinary synchronizations between the automaton, so from the semantical point of view those do not differ from ordinary transitions. The theorem also allows building the reduced state space of a system $TA_1 \parallel \ldots \parallel TA_n$ in a compositional manner by first building the reduced state spaces of the individual components TA_i and afterwards combining those with parallel operator.

With the above results, we can now conclude the following:

Corrollary 4 *Let* TA_1, TA_2 *and TA be weakly constrained timed automata.*

1 *The language accepted by TA is nonempty if and only if* RS_{TA} *contains a cycle which is reachable from the initial state, and contains at least one* 1*-transition, and a control state component* $s \in F$.

2 *If TA is built from several parallel components, like* TA_1 *and* TA_2, *then the emptiness of TA can be checked by observing the existence of such a cycle from* $RS_{TA_1} \parallel RS_{TA_2}$.

The existence of a 1-transition in the cycle is required in order to ensure that the corresponding timed word is progressive.

These results show how the verification of dense time properties of a weakly constrained timed automaton can be reduced to cycle detection

from a finite discrete structure which is suitable for algorithmic verification. However, some further optimizations can be achieved by noticing that all the state information except the fact whether a state is accepting or not, is actually superfluous for the verification task. So, the structure that is needed greatly resembles the labeled transition systems that are used as semantical models for process algebra descriptions. Actually, our structure, the reduced state space, can be seen as a labeled transition system where the states corresponding to acceptance states are augmented with atomic propositions f.

Next we will rephrase a result concerning the truth of Linear temporal logics preserving reductions for labeled transition systems where states can be augmented with a set of atomic propositions (Kaivola, 1996). Let us call this structure *Augmented labeled transition system.*

In the theorem, the hide-operator well known in process algebra is also used. The semantics of hide is that it simply replaced all the listed transition labels with τ -labels, denoting an internal action. The theorem also refers to the CFFD (Chaos Free Failures Divergences) semantical model (Valmari and Tienari, 1995), which is a semantics inducing an equivalence relation and the related pre-order between labeled transition systems. It is a variant of the Failures Divergences model well known from the CSP community.

Theorem 5 *Let* $ALTS_i = (S_i, A_i, R_i, s_{0_i}, L_i)$ *be alts's for* $i \in \{1,2\}$, *where* L_i *is a mapping from* S_i *to the set of atomic propositions true in that state.*

1 *If* $ALTS_1$ *and* $ALTS_2$ *are CFFD-equivalent, then for every formula* φ *of Linear Temporal Logic without 'nexttime'-operator (LTL' in short):* $ALTS_1 \models \varphi$ *if and only if* $ALTS_2 \models \varphi$.

2 *If* φ *is a formula of LTL' that does not refer to actions in the set* A', *then:* $ALTS_1 \models \varphi$ *if and only if* $(\textbf{hide } A' \textbf{ in } ALTS_1) \models \varphi$.

3 *Let* A'_1 *and* A'_2 *be such that* $A'_1 \cap A_2 = \emptyset$ *and* $A'_2 \cap A_1 = \emptyset$, *and furthermore let* φ *be such a LTL' formula that does not refer to actions in* $A'_1 \cup A'_2$. *Now* $(\textbf{hide } A'_1 \cup A'_2 \textbf{ in } (ALTS_1 \,||\, ALTS_2)) \models \varphi$ *if and only if* $(\textbf{hide } A'_1 \textbf{ in } ALTS_1) \,||\, (\textbf{hide } A'_2 \textbf{ in } ALTS_2) \models \varphi$.

4 *If* $ALTS_1 \sqsubseteq_{\text{CFFD}} ALTS_2$ *then for a LTL' formula* φ *if* $ALTS_2 \models \varphi$ *then also* $ALTS_1 \models \varphi$.

This result thus covers the general case with a finite set of state propositions, but we need only one of them, *f,* which is true for the acceptance states only.

Since the nonexistence of an acceptance cycle can be expressed in LTL' with the negation of the formula $\Box\Diamond f \wedge \Box\Diamond executed(1)$, we can use the above result in order to reduce the size of the structure before the cycle detection. Since the property we are verifying refers only to

atomic proposition f and to the transition label 1, it is possible to hide all the other labels. Theorem 3 allows the compositional construction of reduced state space, and because of part 3 of Theorem 5, hiding can be 'pushed' inside the components for the transitions labels that are internal to one component. Then before making the composition, thanks to part 1 of Theorem 5, these components may be minimized with respect to an equivalence relation that is stronger or equal to the CFFD, for example one suitable is the version of weak bisimulation equivalence that takes into account the divergence information. The next corollary summarizes this, as well as the fact that a CFFD-abstraction of a reduced state space can be used to guarantee the nonexistence of an acceptance cycle.

Corrollary 6 *Let TA and $TA_{\neg\varphi}$ be weakly constrained timed automata, where the latter corresponds to negation of the property φ, and let RA_{TA} and $RA_{TA_{\neg\varphi}}$ be the reduced state spaces of the above. Furthermore let RS_{min} be an augmented labeled transition system which is CFFD-equivalent with* hide $A_{TA} \cup A_{TA_{\neg\varphi}}$ in $RS_{TA} \,||\, RS_{TA_{\neg\varphi}}$. *Now the following are equivalent:*

1. *TA satisfies the property φ,*
2. *$RS_{TA} \,||\, RS_{TA_{\neg\varphi}}$ does not have any accepting cycles containing 1 transitions,*
3. *RS_{min} does not have any accepting cycles containing 1 transitions.*

Furthermore, let RS' be an augmented labeled transition system, for which the relation $RS_{min} \sqsubseteq_{\mathrm{CFFD}} RS'$ holds. If RS' does not have any accepting cycles containing 1 transitions then TA satisfies the property φ.

The last part of the corollary is an interesting consequence of part 4 of Theorem 5. It allows the possibility of replacing some components in the reduced state space with ones that are 'bigger' with respect to CFFD pre-order, and if the reduced state space with this 'bigger' version (often called *abstraction*) does not contain cycles, it is guaranteed that the concrete system does not contain a cycle either.

As the following theorem shows, the nonexistence of an acceptance cycle can be verified when the time progress transitions are hidden.

Theorem 7 *Let TA and $TA_{\neg\varphi}$ be weakly constrained timed automata, and RS' =* hide $A_{TA} \cup A_{TA_{\neg\varphi}} \cup \{1\}$ in $RS_{TA} \,||\, RS_{TA_{\neg\varphi}}$. *If RS' does not contain any accepting cycles then TA satisfies the property φ.*

Thus, the cycle detection might at first be done for a reduced state space that has no visible transitions at all, and only if a cycle is found, then the time progress transition 1 is unhidden and the existence of the cycle is re-checked.

If only some of the system components actually refer to the progress of time, some further optimization can be achieved. This is formalized with the following theorem.

Theorem 8 *Let* TA_1 *be an arbitrary timed automaton,* TA_2 *a timed automaton that allows time to progress in every state, and* RS_{TA_1}, RS_{TA_2} *the corresponding reduced state spaces. Let us denote with* RS'_{TA_2} *the alts which is obtained from* RS_{TA_2} *by removing all the* 1 *transitions. Now* $RS_{TA_1} \parallel RS_{TA_2} =_{\mathrm{CFFD}} RS_{TA_1} \parallel RS'_{TA_2}$.

4. FISCHER'S PROTOCOL

Fischer's mutual exclusion protocol uses only one shared variable in guaranteeing the mutual exclusion, but it requires correct timing from the processes trying to enter their critical sections.

Every client of the protocol is given a unique identifier *id* which is a natural number greater than zero. When a client wants to enter the critical section, it first waits until the value of the shared variable is zero which indicates that nobody is currently in the critical section. When detecting that, the client writes its own *id* to the shared variable. This operation has to be done within T time units from the moment when the value of the shared variable was noted as zero. After that, the client waits for T' time units, where $T' > T$. If the shared variable still contains the *id* of the client, it is allowed to enter the critical section. In other cases, the mutex-protocol is started from the beginning.

Every process as well as the memory of the system are modeled as timed automata. The automata P_i and *Mem,* modeling process i and the shared memory are defined in Figure 1. As usual, we assume that all the states in the automata describing the system are accepting.

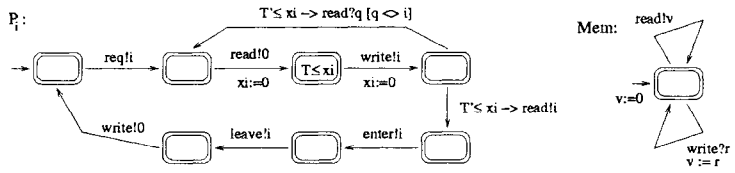

Figure 1. Fischer's protocol

A timed automaton describing the behavior of the system is obtained with the parallel composition $(P_1 \,|||\, \ldots \,|||\, P_n) \parallel Mem$, where $|||$ denotes the full interleaving operator.

The negation of the mutual exclusion property, which states that there is maximally one process at a time in the critical section, is captured by the automaton $\overline{Mutex}$ which is defined in Figure 2.

The protocol satisfies the mutual exclusion property if we can show that the intersection of the languages defined by $(P_1 \,|||\, \ldots \,|||\, P_n) \parallel Mem$ and $\overline{Mutex}$ is empty.

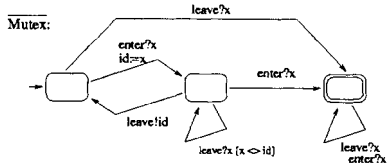

Figure 2. Automaton defining the negation of the mutual exclusion property

In order to do the verification we produced the transition systems $RS_{\overline{Mutex}}$, RS_{Mem} and RS_{P_i} for every $i \in \{1, \ldots, n\}$. The results with parameter values $T = 1$, $T' = 2$ and $n = 2$ are shown in Figure 3.

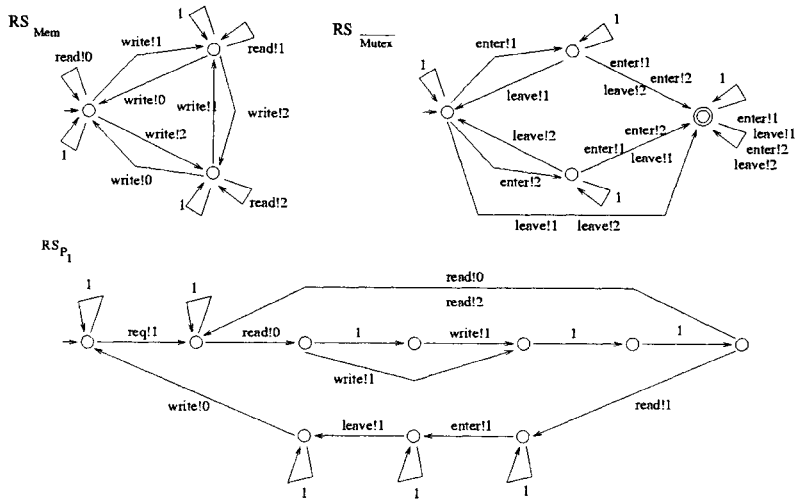

Figure 3. Reduced state spaces for Fischer's protocol

Then the next step was building the transition systems

$$System := red(red(\text{hide } read, write, req, 1 \text{ in } red(RS_{P_1}) \,|||\, \ldots \,|||\, red(RS_{P_n})) \,||\, red(\text{hide } 1 \text{ in } RS_{Mem}))$$

$$Test := red(\text{hide } enter, leave \text{ in } System \,||\, red(\text{hide } 1 \text{ in } RS_{\overline{Mutex}})),$$

where *red* denotes a reduction procedure that preserves CFFD-equivalence. Transition systems *System* and *Test* with the example parameters are shown in Figure 4.

Now because of Corollary 6 and Theorem 8 it follows that if *Test* does not have any accepting cycles then the intersection of the languages defined by $(P_1 \,|||\, \ldots \,|||\, P_n) \,||\, Mem$ and $\overline{Mutex}$ is empty, and thus the protocol satisfies the mutual exclusion property. Since all the automata

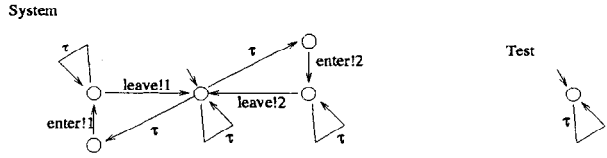

Figure 4. Transition systems *System* and *Test* in the example case

are weakly constrained, the verification results hold for the dense time domain.

We verified the system with several parameter value combinations. The ARA (Advanced Reachability Analysis) toolset (Valmari et al., 1993) was used for the verification. The sizes of the largest transition systems that were encountered during the verification are shown in Table 1. For the compositional style of building the state space it is often the case that not the final result, but some intermediate step is the one that produces the largest transition system, thus the one that restricts the size of the systems that the tool can handle. So, for the compositional method, the largest intermediate result is shown in the table. Besides the number of client processes, the time parameter values have a great impaction the size of the verification model. This fact is also reflected in Table 1.

n	*T*	*T'*	compositional	noncompositional
3	0	1	294	415
3	1	2	496	777
3	4	5	1882	3351
4	0	1	1417	1899
4	1	2	3013	4559
4	4	5	19340	33399
5	0	1	6901	7205
5	1	2	18838	27769
5	4	5	200011	-
6	0	1	34124	43854
6	1	2	120254	-
7	0	1	172115	-
8	0	1	156564	-

Table 1. Sizes of the reduced state spaces with the various parameter combinations. Note that the 8 client case was done with an optimized version of the protocol clients, where the *enter, req* and *leave* transitions were not used, but instead just the information whether the client is in its critical section or not.

Let us now demonstrate what happens if the property to be verified does not hold. Consider the protocol in case of two clients and parameter values $T = 1$, and $T' = 2$ for P_1 but $T = T' = 1$ for P_2. The transition systems *System* and *Test* are shown in Figure 5.

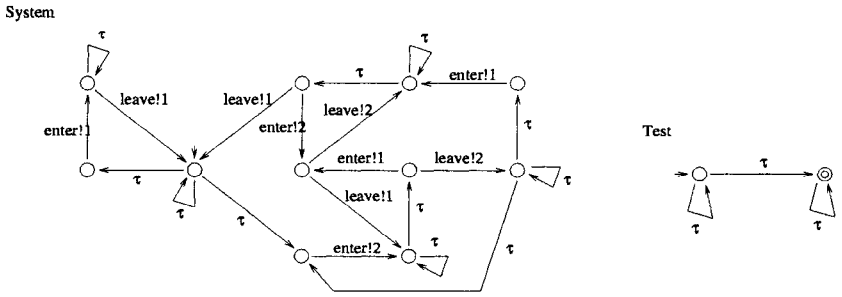

Figure 5. *System* and *Test* in the case of an erroneous parameter value in the second entity

As is seen, the *Test* now contains an accepting cycle, so the property might not hold. The next step is to built *System'* and *Test'* which are just as the above two, except the transition label 1 is not hidden. The result *Test',* which is shown in Figure 6, indeed contains an accepting cycle, with a l-transition, so Corollary 6 now says that the this version of the protocol does not guarantee the mutual exclusion.

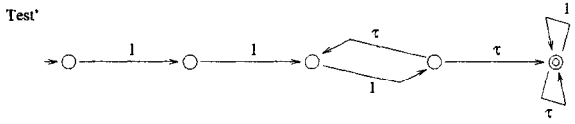

Figure 6. *Test'* showing the existence of an accepting cycle with progress of time

It is impossible to see the reason for the error from *Test',* and the only way to produce a useful counter-example is trying to preserve more transition information in the transition system from which the cycle is to be detected.

In comparison with other reported studies on Fischer's protocol, our compositional method performs quite well. The Discrete Time SPIN (Bošnački and Damms, 1998) was able to do the verification up to 6 client processes, and the Real Time Spin (Tripakis and Courcoubetis, 1996) with 4 clients and UPPAAL with 8 clients. The tool Kronos has been reported to manage with 5 clients (Daws et al., 1996), and UPPAAL with 8 (Larsen et al., 1995). With the convex-hull over approximation tech-

nique (Daws and Tripakis, 1998) both UPPAAL and Kronos can handle the protocol with 9 clients. In the above case studies, the timing parameters T and T' used were not reported, but since those methods, except Discrete Time SPIN, rely on region or zone techniques, it is presumable that the results are not as sensitive to the values of time parameters as ours.

5. BOUNDED RETRANSMISSION PROTOCOL

The bounded retransmission protocol, or BRP in short (D'Argenio et al., 1997), is used to implement a reliable file transfer service using an unreliable point-to-point data link, such as a telephone line. The underlying data link offers unreliable delivery of small, fixed size data *packets*. The size of the packets is far smaller than a typical file which has to be transferred. Thus, the file has to be fragmented into *frames,* which are smaller than a data packet of the link.

The formal specification of the protocol consists of four timed automata: *Sender, Receiver,* Ch_{SR}, Ch_{RS} where the latter two describe the behavior of the underlying channel. The description of the entire system is parallel composition of the automata.

Due to the restricted length of the paper, only the verification result is reported here, and we suggest that reader see the full version of the paper (M. Luukkainen, 2001) to get a more detailed view of the protocol and how the verification was done.

The parameter describing the length of the file in the *Sender* automaton turned out to be problematic from the verification point of view. In fact, there has to be an upper limit in the length of the transmitted file in order to make *Sender* finite stated. And even if a limit is set, it has a great impact on the size of the generated transition systems. Therefore, we used an abstraction *Sender'* of the sender automaton. In *Sender'* the actual length of the transferred file is not remembered, but instead it is 'guessed' while sending the file.

Let us denote the sender with k as the maximum length of files as $Sender_k$. Since for all values of k the reduced state space RS_{Sender_k} of the actual sender automaton is CFFD refinement of the transition system $RS_{Sender'}$ of the abstraction, i.e. $RS_{Sender_k} \sqsubseteq_{\mathrm{CFFD}} RS_{Sender'}$, the Corollary 6 guarantees, that if there are no acceptance cycles in the system with the abstract sender, the property holds for the actual version of the system. So, with the use of an abstraction of the sender, we now gain two things. Firstly there is a substantial saving in the size of the generated transition systems, since already the size of $Sender_3$ is

greater than that of the abstraction. Even if the effect of increasing k appears to be linear, see the Table 2, it is advantageous to eliminate this source of state explosion. The second, even more impressive advantage is that the verification generalizes to all the possible file lengths.

	2	4	6	8	10	Abstraction
Sender	51	107	163	219	275	65
Model	1156	2393	3628	4864	6100	1433

Table 2. Effect of the maximum file length to the size of sender and to the verification model in case the channel delay is 1 and number of retransmissions is 1

The correctness requirements of the protocol are twofold. The automaton defining *the correspondence of indications given to clients of the protocol* was quite straightforward to construct. The verification itself was then done in the same way as in the case of Fischer's protocol.

From the verification point of view, the other requirement *the frames are received in correct order, and no frames are dropped if an error is not indicated* was more problematic. To verify this property, the theory of *data independence* (Wolper, 1986) was exploited. The data independence guarantees that, if a transfer protocol does not alter or use in its logic the user data, it is enough to consider just a small number of different types of user data frames, even if the domain of the frames is extremely large. So, with the help of this we formulated the corresponding property automaton and carried out the verification in the same way as was reported for Fischer's protocol.

The verification was done with several choices of values for the parameters D_{min}, D_{max}, and MAX where the first two define the transmission delay of a frame, and the last describes the maximum number of retransmission trials for a frame. Table 3 summarizes the sizes of the largest reduced states spaces during the verification.

D_{min}	D_{max}	MAX	verifying indications	verifying data transfer
0	1	1	1433	1993
0	1	3	5415	7411
1	2	1	2108	4499
1	2	3	8915	11693
2	3	1	2958	3748
2	3	3	12967	16378

Table 3. Sizes of the largest encountered reduced state spaces with the various parameter combinations.

In D'Argenio et al., 1997 verification of the protocol with UPPAAL was reported. The main difference compared to our results is that in the study of D'Argenio et al., 1997 the verification was done only for files of length 1, 2 and 3 frames, and the correctness of data transfer was not verified. In the UPPAAL study the channel delay was fixed to one, and as in our case, the maximum number of retransmissions between 1 and 3 were considered.

In order to compare the efficiency of our method versus UPPAAL, we ran our verification with a Sender process where the maximum length of transmitted files was fixed to 3. With the number of retransmissions 1, 2 and 3 the running times with our method were 2, 4 and 13 seconds and with UPPAAL the corresponding measures were 61, 156 and 340 seconds. Our method also outperforms the Discrete Time SPIN verification reported in Bošnački and Damms, 1998.

6. CONCLUSIONS

In the article we showed how the untimed process algebra theory and tools can be used in verifying a category of dense time properties from a restricted class of timed automata. The work was based on the known result that by restricting ourselves to a class of timed automata where only the operators $\leq$, $=$, and $\geq$ are used in the clock comparisons, most of the interesting dense time properties are verifiable using discrete time methods. Those results were refined in order to fit the process algebraic context, and furthermore, some reduction strategies based on behavior equivalences, abstraction and compositionality were indicated. It is in particular the possibility for the compositional generation of state spaces that makes the process algebraic method efficient for the timed systems, as well.

The experimental results obtained from the verification of Fischer's mutual exclusion protocol and Bounded retransmission protocol indicated that the efficiency of our method is of the same magnitude as for the other available timed methods. Especially in the BRP case study our method clearly worked faster than UPPAAL in the study reported in D'Argenio et al., 1997. An interesting point in our BRP study was the possibility of using an abstract version of the sender process that allowed us to conduct the verification for arbitrary long files. Thus, with our method, we were able to obtain a more complete verification result than the UPPAAL, since there the maximum length of files had a fixed upper bound of just 3 frames.

REFERENCES

Alur, R. and Dill, D. (1994). A Theory of Timed Automata. *Theoretical Computer Science,* 126:183–235.

Bošnački, D. (1999). Digitization of Timed Automata. In *Fourth International Workshop on Formal Methods for Industrial Critical Systems.*

Bošnački, D. and Damms, D. (1998). Integrating Real-Time in Spin: a Prototype Implementation. In *Proceedings of IFIP Joint Conference on Formal Description Techniques and Protocol Specification, Testing and Verification,* pages 423–439. Kluwer Academic Publishers.

D'Argenio, P., Katoen, J.-P., Ruys, T., and Tretmans, J. (1997). The Bounded Retransmission Protocol Must Be in Time. In *Third International Workshop of Tools and Algorithms for the Construction and Analysis of Systems,* number 1217 in Lecture Notes in Computer Science, pages 416–431. Springer-Verlag.

Daws, C., Olivero, A., Tripakis, S., and Yovine, S. (1996). The Tool KRONOS. In *Hybrid Systems III,* number 1066 in Lecture Notes in Computer Science, pages 208–219. Springer-Verlag.

Daws, C. and Tripakis, S. (1998). Model Checking of Timed Reachability Properties Using Abstraction. In *Fourth International Workshop of Tools and Algorithms for the Construction and Analysis of Systems,* Lecture Notes in Computer Science. Springer-Verlag.

Henzinger, T., Manna, Z., and Pnueli, A. (1992). What Good are Digital Clocks? In *Proceedings of the 20th International Conference on Automata, Languages and Programming,* number 623 in Lecture Notes in Computer Science, pages 545–558. Springer-Verlag.

Kaivola, R. (1996). *Equivalences, Preorders and Compositional Verification for Linear Time Temporal Logic and Concurrent Systems.* Report A-1996-1. PhD Thesis, University of Helsinki, Department of Computer Science.

Larsen, K. G., Pettersson, P., and Yi, W. (1995). Model-Checking for Real-Time Systems. In *Proc. of Fundamentals of Computation Theory,* number 965 in Lecture Notes in Computer Science, pages 62–88.

M. Luukkainen (2001). Verification of dense time properties using theories of untimed process algebra. www.cs.helsinki.fi/u/mluukkai/full.ps.

Tripakis, S. and Courcoubetis, C. (1996). Extending Promela and Spin with Real Time. In *Second International Workshop of Tools* and *Algorithms for the Construction and Analysis of Systems,* Lecture Notes in Computer Science. Springer-Verlag.

Valmari, A., Kemppainen, J., Clegg, M., and Levanto, M. (1993). Putting Advanced Reachability Analysis Techniques Together: the "ARA" Tool. In *Proceedings of Formal Methods Europe,* number 670 in Lecture Notes in Computer Science, pages 597–616. Springer-Verlag.

Valmari, A. and Tienari, M. (1995). Compositional Failure-Based Semantic Models for Basic LOTOS. *Formal Aspects of Computing,* 7:440–468.

Wolper, P. (1986). Expressing Interesting Properties of Programs in Propositional Temporal Logic. In *Proceedings of the 13th ACM Symposium on Principles of Programming Languages,* pages 184–193.