Timed automata and the theory of real numbers

Hubert Comon Yan Jurski

LSV, ENS Cachan, 94235 Cachan cedex, France
{comon,jurski}@lsv.ens-cachan.fr
http://www.lsv.ens-cachan.fr/∼comon
Tel: 01 47 40 24 30. Fax: 01 47 40 24 64

Abstract. A configuration of a timed automaton is given by a control state and finitely many clock (real) values. We show here that the binary reachability relation between configurations of a timed automaton is definable in an additive theory of real numbers, which is decidable. This result implies the decidability of model checking for some properties which cannot be expressed in timed temporal logics and provide with alternative proofs of some known decidable properties. Our proof relies on two intermediate results: 1. Every timed automaton can be effectively emulated by a timed automaton which does not contain nested loops. 2. The binary reachability relation for counter automata without nested loops (called here *flat automata*) is expressible in the additive theory of integers (resp. real numbers). The second result can be derived from [10].

1 Introduction

Timed automata have been introduced in [4] to model real time systems and became quickly a standard. They roughly consist in adding to finite state automata a finite number of clocks which grow at the same speed. Each transition comes together with some clock resets and an enabling condition, whose satisfaction depends on the current clock values. Temporal properties of real time systems have been expressed and studied through temporal logics such as TPTL [7], MITL [6], TCTL [2, 15, 19],timed μ-calculi [15, 16]. These logics are in general undecidable, with the notable exception of MITL. On the other hand, the model-checking is decidable for the (real-time) branching time logics, though hard in general.

Timed models are harder than untimed ones since they can be seen as infinite state systems in which every configuration consists of a pair of a control state (out of a finite set) and a vector of real clock values. Reasoning about possible clocks values in each state is the core of the difficulty. In this paper, we adopt the following point of view: infinite sets of configurations can be finitely described using *constraints*. For instance, "$(q, x \geq y + z)$" is the set of configurations in control state q and such that the clock x is larger than the sum of clocks y and z. This point of view is not new, as the *regions* of [2], which are used in a crucial way in the verification algorithm, are indeed a representation of sets of configurations indeed. Here, we go one step further: we express not only sets of configurations, but also *relations* between configurations in a (decidable) constraint system. Then temporal properties of the model are described through the binary reachability relation $\xrightarrow{*}$ relating clock values, which is expressible in the constraint system. Since we may always assume that there is a clock τ which is never reset by the automaton (and hence is a witness of the total elapsed time), we may express for instance some delay conditions such as "d is a delay between q and q'" as a constraint: $\exists \boldsymbol{x}, \boldsymbol{x'}, \tau.(q, \boldsymbol{x}, \tau) \xrightarrow{*} (q', \boldsymbol{x'}, \tau + d)$. Now it is possible to analyse delays between some events such as finding minimal or maximal delays. There are already algorithms which find such extremal delays [12], but we may also decide properties such as: "the delay between event a and event b is never larger than twice the delay between event a' and event b'" (which is, up to our knowledge, a new decidability result). More generally, our main result is that the binary reachability relation between clocks values, which is defined by a timed automaton, is effectively expressible in the additive theory of real numbers. Since the additive theory of real numbers is decidable, any property which can be expressed in this theory using the reachability relation, can be decided. In particular, we can compute reachable configurations from a definable set of configurations as well as the set of configurations from which we can reach a definable set. Hence we have forward and backward model-checking algorithms of safety properties as simple instances of our result. But we may also check properties which express relations between the original and final clock values. Also, some parametric verification is possible as we may keep free variables in the description of original and final configurations: for safety properties, the results of [18] can be derived from our main result. Finally, we can handle more general models than timed automata: transitions may be guarded by arbitrary first-order formulas over clocks, provided that such transitions can only be fired a bounded number of times.

On the negative side, not all timed temporal properties can be expressed in the first-order theory of $\xrightarrow{*}$. Typically, unavoidability is not expressible. This is not surprising since our logic is decidable, whereas the timed temporal logics are not in general.

Our main result is proved in two steps: first we show that any timed automaton can be emulated by an automaton without nested loops, hereafter called *flat automaton*. The notion of emulation will be precised, but keep only in mind that it preserves the reachability relation. Hence, in some sense, timed automata with a star height n are not more expressive than timed automata with star height 1. (This is not true, of course, if we consider the accepted language instead of the reachability relation as an equivalence on automata). The second step consists of applying one of our former results, which shows that the reachability relation is effectively expressible in the additive theory of real (resp. integer) numbers for flat counter automata [10]. We go from timed automata to automata with counters using an encoding due to L. Fribourg [13].

The emulation result itself is proved in three steps: first we define an equivalence relation on transition sequences, which we show to be right compatible and of finite index. This is similar to a region construction, though the equivalence is rather on pairs of configurations than on configurations. Second, we show some commutation properties of equivalent transition sequences: roughly, equivalent transition sequences can be performed in any order, without affecting the reachability relation. The third (and last) step consists in using combinatorial arguments on words and proving that there is a flat automaton whose language contains a set of representatives for the congruence generated by the commutation properties. (This result can be stated as a formal language property which is independent from the rest of the paper). This provides with a mechanical way to choose which transition sequence to commute with another: the representatives of the congruence classes are the normal forms w.r.t. a regular string rewriting system.

From this proof, we can also derive some other decidability results. For instance, we can decide whether a sequence of transitions can be iterated.

In section 2, we recall the basic definitions of timed automata and we introduce our constraint system. Next, we sketch the proof of the emulation result in section 3 and derive in section 4 the definability of the reachability relation. In section 5, we show some examples of temporal properties which can be expressed in the theory of real numbers. In particular we give examples showing the expressiveness of the binary reachability relation. We conclude in section 6. Many constructions are only sketched in this abstract paper. More details can be found in [11].

2 Timed Automata

We start with a classical notion of timed automaton, which includes invariants in the states and guarded transitions. The syntax and semantics of timed transition systems we use here is not important: we can switch from the following definitions to others (such as [5]) without changing our main result. The events and the accepted language are also irrelevant here, as we are interested in reachability.

2.1 Syntax and semantics

Let B be a finite set of real numbers (we will assume later that these constants are in $\mathbb{Z}$) and C a finite number of variables called *clocks.* $\Phi(B, C)$ is the set of conjunctions[1] of atomic formulas of the form $x \leq c, x \geq c, x < c, x > c, x = c$ where $x \in C$ and $c \in B$.

Definition 1 ([1]). *A* timed automaton *is a tuple* $< \Sigma, Q, Q_0, C, I, E >$ *where*

- Σ *is a finite alphabet*
- Q *is a finite set of states (and* $Q_0 \subseteq Q$ *is a set of start states, irrelevant here)*

[1] Having arbitrary Boolean combination does not increase the expressive power. We choose to consider conjunctions only since they guarantee a convexity property for the invariant constraints.

– *C is a finite set of clocks*
– *I is a mapping from Q to $\Phi(B, C)$ (the* invariant *associated with each state).*
– *$E \subseteq Q \times Q \times \Sigma \times 2^C \times \Phi(B, C)$ gives the set of transitions. In each transition $< q, q', a, \lambda, \phi >$, λ is a set of* clock resets *and ϕ is a* clock constraint.

A *configuration* of the automaton is a pair $(q, \boldsymbol{V})$ where $q \in Q$ and $\boldsymbol{V} \in \mathbb{R}_+^{|C|}$ is a clock value. There is a *move* of a timed automaton $\mathcal{A}$ from a configuration $(q, \boldsymbol{V})$ to $(q', \boldsymbol{V'})$, which we write $(q, \boldsymbol{V}) \underset{\mathcal{A}}{\longrightarrow} (q', \boldsymbol{V'})$, iff

– Either $q = q'$ and $\boldsymbol{V} \models I(q), \boldsymbol{V'} \models I(q)$ and there is a positive real number t such that, for every component i, $v'_i = v_i + t$.
– Or else there is a transition $< q, q', a, \lambda, \phi >$ and a positive real number t such that $\boldsymbol{V} \models \phi$ and for every component i, either $v_i \in \lambda$ and then $v'_i = t$ or else $v_i \notin \lambda$ and $v'_i = v_i + t$. Moreover, $\boldsymbol{V'} \models I(q')$.

$\overset{*}{\underset{\mathcal{A}}{\longrightarrow}}$ is the reflexive transitive closure of $\underset{\mathcal{A}}{\longrightarrow}$. We also write $\xrightarrow[q_1, q_2]{*} \subseteq \mathbb{R}^{|C|} \times \mathbb{R}^{|C|}$ the relation on clocks vectors defined by $\boldsymbol{V} \xrightarrow[q_1, q_2]{*} \boldsymbol{V'}$ iff $(q_1, \boldsymbol{V}) \overset{*}{\underset{\mathcal{A}}{\longrightarrow}} (q_2, \boldsymbol{V'})$. We will always assume without loss of generality that there is a clock τ which is never reset.

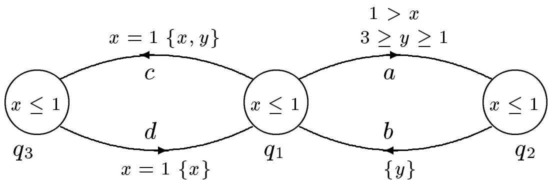

Fig. 1. A timed automaton

Example 1. An example of a timed automaton is displayed on figure 1. As usual, invariants are written in the states and enabling conditions label the edges. The variables which are reset by a transition are written inside brackets. We assume that there are three clocks x, y, τ.

If we consider for instance transitions c, d only, we can express $\xrightarrow[q_1, q_1]{*}$ using the formula:

$$\begin{aligned} \exists \tau_1, x_1, y_1, t_1, t_2.\ & \tau_1 = \tau + t_1 \wedge x_1 = x + t_1 \wedge y_1 = y + t_1 \wedge x_1 \leq 1 \\ & \wedge \exists n. \tau' = \tau_1 + 2n - 1 + t_2 \wedge x' = t_2 \wedge x' \leq 1 \wedge y' = 1 + t_2 \end{aligned}$$

t_1 is the time spent before the first transition c is fired and is specified on the first line. τ_1, x_1, y_1 are the values of the clocks at that date. Then n is the number of times the loop cd is executed and t_2 is the time spent in q_1 after the last transition d has been fired.

This is typically what we will get from our formula computation.

A *flat automaton* is a timed automaton which does not contain nested loops: for every state q there is at most one non-empty path from q to itself.

Example 2. The automaton of figure 1 is not flat. If we remove any of the four transitions, we have a flat automaton.

2.2 Emulation

Definition 2. *A timed automaton $\mathcal{A}'$* emulates *$\mathcal{A}$ if there is a mapping ϕ from the set of states of $\mathcal{A}'$ into the set of states of $\mathcal{A}$ such that, for every states q, q' of $\mathcal{A}$ and every clock vectors $\boldsymbol{V}, \boldsymbol{V'}$, $(q, \boldsymbol{V}) \xrightarrow[\mathcal{A}]{*} (q', \boldsymbol{V'})$ iff there are states $q_1 \in \phi^{-1}(q)$, $q_1' \in \phi^{-1}(q')$ such that $(q_1, \boldsymbol{V}) \xrightarrow[\mathcal{A}']{*} (q_1', \boldsymbol{V'})$.*

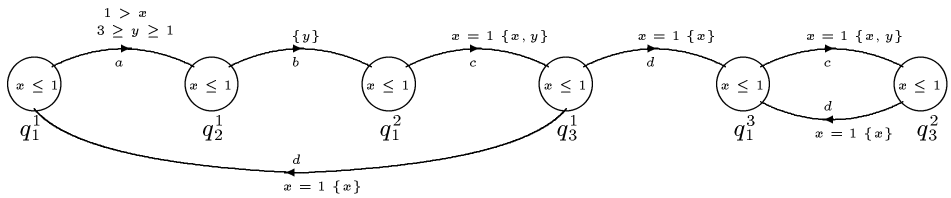

Fig. 2. An emulation automaton

Example 3. The automaton of figure 2 emulates the automaton of figure 1. The states q_i^j are mapped to q_i. It is a flat automaton. It is not straightforward that this automaton indeed emulates the original one. Note for instance that the possible event sequences are different as *abcdcdabcd* is a possible sequence in the automaton of figure 1 and is not a possible sequence in the automaton of figure 2. However, this sequence yields the same binary relation between configurations as the sequence *abcdabcdcd* which is possible in the automaton of figure 2.

The automaton of figure 2 is typically what we want to compute from the automaton of figure 1.

Lemma 1. *If $\mathcal{A}'$ emulates $\mathcal{A}$ then* $\xrightarrow[q_1,q_2]{*} = \bigcup_{q_1' \in \phi^{-1}(q_1), q_2' \in \phi^{-1}(q_2)} \xrightarrow[q_1',q_2']{*}$

Hence, as far as $\xrightarrow{*}$ is concerned, we may consider any automaton emulating $\mathcal{A}$ instead of $\mathcal{A}$ itself.

2.3 The additive theory of real numbers

The theory $\mathcal{T}$ we consider here is defined as follows. Terms are built from variables, the constants $0, 1$ and the function symbol $+$. Formulas are built using first-order quantifiers and the usual logical connectives on atomic formulas which are either equations $u = v$ between terms or predicates $Int(u)$ where u is a term.

The domain of interpretation of such formulas is the set of non-negative real numbers, with the usual interpretation of function symbols. Int is the set of natural numbers.

This theory can be encoded in S1S using (infinite) binary representation of real numbers. Hence it is a decidable theory.[2]

Example 4. The formula of example 1 is a formula of this theory.

[2] We do not know the complexity of this theory, nor of the fragment of $\mathcal{T}$ which is used here.

3 Every timed automaton can be emulated by a flat timed automaton

The automaton of figure 1 is not flat because there are two loops ab and cd on state q_1. If the order of the two loops was irrelevant to the reachability relation, we could switch them and assume that all sequences ab are performed before sequences cd. Then we would get a flat automaton, first considering the loop ab and then the loop cd. However, in this example, we cannot switch the two sequences because, for instance, $abcd$ and $cdab$ do not induce the same relations on the initial values of the clocks. Then, the question is: when can we switch two sequences of transitions w and w' without altering the reachability relation? Let us look first at some necessary conditions.

If w and w' do not induce the same relations on initial clock values, then their order is relevant since, for instance w may occur after some other transition sequence, whereas w' cannot. This is the case in our example: ab and cd do not induce the same relations on initial clock values and ab cannot occur after another ab, whereas cd can occur after ab. $ababcd$ is impossible and $abcdab$ is possible. Hence a first necessary condition is that w and w' induce the same constraints on initial clock values.

Similarly, w and w' should enable the same transitions: whereas w or w' has been executed last should not be relevant for further transitions. This means that w and w' should induce the same constraints on final clock values, or at least constraints that can be met by the same enabling conditions of further transitions.

There are further necessary conditions for two transition sequences to commute.

Example 5. For instance consider the automaton displayed on figure 3. On this

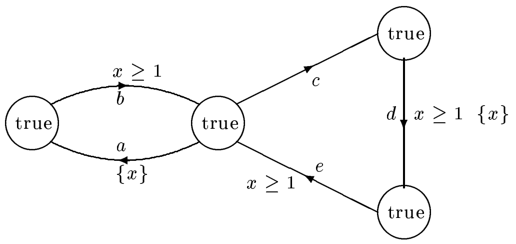

Fig. 3. Another example of non-commuting loops

example, let $w = ab$ and $w' = cde$. Executing ww' yields a constraint $x' - x \geq \tau' - \tau - 1$ on the final and initial x's values whereas $w'w$ yields a strictly weaker constraint $x' - x \geq \tau' - \tau - 2$. (To see this, start with $\tau = 1$ and $x = 0$. $w'w$ may yield a configuration in which $\tau' = 4$ and $x' = 1$, which is not reachable using ww'). On the other hand both w and w' induce the same constraints on initial clock values and on final clock values. Hence another necessary condition for switching w and w' is that they induce the same discrepancy between $x - x'$ and $\tau - \tau'$.

We will see that, very roughly, these three necessary conditions are also sufficient.

We are going to define a right compatible equivalence $\sim$ on transition sequences such that, in particular, the above situations cannot occur when $w \sim w'$. $\sim$ will be of finite index. Hence we will be able to split the states according to its equivalence classes, ensuring that two sequences starting from the initial state and which have the same final extended state can be switched, without changing the reachability relation.

3.1 A right compatible equivalence on transition sequences

This is the analog of *regions*, considering pairs of configurations instead of single configurations. Roughly, in the regions construction, two configurations $(q, \boldsymbol{v})$ and $(q, \boldsymbol{v'})$ are considered as equivalent, if they satisfy the same constraints $x \geq y + c$, $x > y + c$ where x, y are clocks and c is a constant (which is bounded by the largest constant of the model). Here, we define a right compatible equivalence on pairs of configurations $(q_1, \boldsymbol{v_1}), (q_2, \boldsymbol{v_2})$ and $(q_1, \boldsymbol{v'_1})$, $(q_2, \boldsymbol{v'_2})$. Two such pairs are equivalent, roughly, if they satisfy the same constraints $x \geq y' + c, x \geq x + c \ldots$, i.e. not only constraints relating clock values at a given time, but also constraints relating clock values before and after a sequence of transitions. The situation is not as simple as in the region case, however. Indeed, the relevant constants c now range over an a priori infinite set.

Let E be the set of transitions and w, w' be transition words. Let moreover ϕ_w be the formula with free variables $\boldsymbol{X}, \boldsymbol{X'}$ which expresses the relationship between clock values before and after w:

$$(q, \boldsymbol{V}) \xrightarrow[\mathcal{A}]{w} (q', \boldsymbol{V'}) \text{ iff } \boldsymbol{V}, \boldsymbol{V'} \models \phi_w$$

Note that this formula is independent of the states q, q' since we gave a different name for each transition of $\mathcal{A}$. Hence, given w there is only one starting and one target state for w. (Actually, q, q' are implicitly "encoded" into w).

As we want to commute sequences of transitions of the same class, we will need to keep a control property in the equivalence relation. $\sim_0$ ensures that two equivalent words are computations between the same states : $w \sim_0 w' \stackrel{\text{def}}{=}$ *the source and target states of* w, w' *are identical.*

The first property we want to keep is the relation between the initial clock values: let $\sim_1$ be defined by ($\#$ is either $\leq$ or $<$; in the following we will only consider $\leq$ for sake of simplicity):

$$w \sim_1 w' \stackrel{\text{def}}{=} \forall x, y \in \boldsymbol{X}, \forall c \in \mathbb{R}, \quad \phi_w \models x \# y + c \text{ iff } \phi_{w'} \models x \# y + c$$

Example 6. Let us consider again the automaton of figure 1 and let S be the transition sequence ab and T be the transition sequence cd. Then $S \not\sim_1 T$. However, we have $ST^n \sim_1 ST^m$ and $T^{n+1} \sim_1 T^{n+k}$ for every n, m, k.

The finite index property of $\sim_1$ will be guaranteed if constants belong to $\mathbb{Z}$ since, if K is the maximal constant occurring the enabling conditions or invariants of the automaton:

Lemma 2. *For every w, x, y, $min\{c \in \mathbb{R} \mid \phi_w \models x \leq y + c\} \in [-K; K] \cup \{+\infty, -\infty\}$.*

However, $\sim_1$ is not right compatible as we may have $w \sim_1 w'$ and, for some transition t, $w \cdot t \not\sim_1 w' \cdot t$. That is because the transition t may introduce some constraint which is backward propagated to the initial values of the clocks. And this propagation does not depend on the initial values of the clocks only. In addition to this first difficulty, if we define a similar relation on the final values of the clocks: $w \sim_2 w' \stackrel{\text{def}}{=} \phi_w \models x' \leq y' + c$ iff $\phi_{w'} \models x' \leq y' + c$, we also loose the finite index property since the analog of lemma 2 does not hold for final values of the clocks: the minimal difference imposed by a sequence of transitions on the final values of the clocks can be arbitrarily large. (Think of a clock which is reset at each transition of w and another clock which is never reset). Hence we have to define $\sim_2$ as an approximation w.r.t the above definition:

$$w \sim_2 w' \stackrel{\text{def}}{=} \forall c \in [-K; K] \cup \{-\infty, +\infty\},\ \phi_w \models x' \leq y' + c \text{ iff } \phi_{w'} \models x' \leq y' + c$$

Example 7. In our example, there are only two classes of sequences in $(S + T)^*$ w.r.t. $\sim_1$: $ST(S+T)^*$ and $T(S+T)^*$. Now, w.r.t. $\sim_2$, sequences are also distinguished according to their last transition: constraints on final clock values depend on whether the last transition is S or T.

It is out of the scope of this extended abstract to show in details how we solve the above mentioned difficulties (see [11] for more details, in particular all definitions and proofs concerning $\sim_i$). The main idea is to define other relations $\sim_i$ which, altogether, will give a right compatible equivalence of finite index. The idea behind $\sim_3$ is to restore the compatibility of $\sim_1$ w.r.t. the right composition. Hence $\sim_3$ anticipates possible relations between initial clock values. More precisely, assume that $w \sim_1 w'$. $w \cdot t$ may induce a new constraint on the initial clock values in the following situation: $\phi_w \models x \leq y' + c_1 \wedge z' \leq u + c_2$ and $t \models y \leq z + c_3$. Then composing w and t, we get $\phi_{w \cdot t} \models \exists y_1, z_1 . x \leq y_1 + c_1 \wedge z_1 \leq u + c_2 \wedge y_1 \leq z_1 + c_3$, which implies $x \leq u + c_1 + c_2 + c_3$, possibly a new relation between initial clock values.

Roughly $w \sim_3 w'$ if the sums $c_1 + c_2$ such that we have such relations are the same for w and for w'. $c_1 + c_2$ may take arbitrary real values. However, we know that $c_1 + c_2 + c_3 \in [-K, K]$, thanks to lemma 2, otherwise $x \leq u + c_1 + c_2 + c_3$ would already be a consequence of ϕ_w. Hence, we only need to consider the sums $c_1 + c_2$ which belong to $[-2K, 2K]$, and we keep a finite index relation.

$\sim_4$ is more complicated. In the spirit, it is the same as $\sim_3$, guaranteeing the compatibility of $\sim_2$. Now, one could think that we have to define $\sim_5, \sim_6$ in order to guarantee the compatibility of the new relations $\sim_2, \sim_3$. Fortunately, this is not the case and, altogether, the union of $\sim_i$ for $i = 0, ..., 4$ is already a right compatible equivalence of finite index.

Still, we have to define more equivalences which take care of e.g. example 5. $\sim_5$ and $\sim_6$ express that the discrepancies between constraints on $x' - u$ and $y' - z$ are the same, up to the constant K. Finally $\sim_7$ is the symmetric of $\sim_3$, roughly reversing the ordering on time. This relation is also necessary for the commutation property (See [11] for more details).

Then we have the expected results, defining $\sim$ as $\bigcup_{i=0}^{7} \sim_i$:

Lemma 3. *$\sim$ is a right compatible equivalence on E^* and $E^*/\sim$ can be effectively computed in time $O(K^{n^2} \times q^2)$ where n is the number of clocks and q the number of states of $\mathcal{A}$.*

Example 8. Continuing example 1, we display on figure 4 the automaton for $\sim$: $u \sim w$ iff u and v are accepted in the same state of this automaton. Let us recall that $S = ab$ and $T = cd$, hence figure 4 is actually an abstraction of the automaton; each transition should be split into two transitions. In addition there should be a trash state for every impossible transition sequence. The complete automaton contains 16 states.

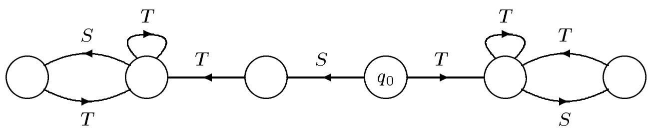

Fig. 4.

3.2 Commutation properties

The first result is that, if we have equivalent sequences of transitions then we can (almost) perform them in any order, without changing the constraint they induce on the clocks values:

Lemma 4. *If $w \sim w' \sim w''$ then $\phi_{w \cdot w' \cdot w''} \models\!\mid \phi_{w' \cdot w \cdot w''}$.*

This lemma shows that, if we have two sequences of transitions w and w' from q to itself and such that $w \sim w'$, then the iteration of both loops i.e. the set of transitions $(w + w')^*$ has the same effect on clocks values as $w^* w'^* (w + \epsilon)$. This shows a flattening operation on regular expressions: $(w + w')^*$ is not flat whereas $w^* w'^* (w + \epsilon)$ is flat.

However, we cannot conclude yet since it is not always possible to compute an automaton emulating $\mathcal{A}$ and such that any two loops on the same state are equivalent for $\sim$. We need a more complex construction which proves that the automaton can be flattened when we have such commutation properties. This is the subject of the next section.

3.3 A formal language property

In this section, we sketch a formal language property which relates commutation properties and flat automata:

Theorem 1. *Let $\sim$ be a congruence of finite index on A^* and let $\approx$ be the least congruence on A^* such that $w \sim w' \sim w''$ implies $w \cdot w' \cdot w'' \approx w' \cdot w \cdot w''$. Then there is a flat automaton (whose all states are final states) which accepts a language containing a set of representatives for $\approx$.*

It is a consequence of the following series of lemmas:

Lemma 5. *Let k be the index of $\sim$. There is a constant $n(k)$ such that any word w of length larger than $n(k)$ can be factorised into $w = w_0 \cdot w_1 \cdot w_2 \cdot w_3 \cdot w_4$ with $w_1 \sim w_2 \sim w_3$.*

This states that any long enough word contains a factor to which the commutation property can be applied. Now, consider the following word rewrite system:

$$R \stackrel{\text{def}}{=} \{uxvyz \ \rightarrow \ vyuxz \mid u >_{lex} v; ux \sim vy \sim z; u, v \in A^{k_1}\}$$

This rewrite system compares lexicographically prefixes of length k_1 of equivalent words and commutes them according to the ordering.

Lemma 6. *R is a terminating rewrite system. The set of irreducible words $NF(R)$ w.r.t R is recognisable. Moreover, if $u \xrightarrow[R]{*} v$ then $u \approx v$.*

Now, we claim that $NF(R)$ is accepted by a flat automaton for some well-chosen k_1. This relies on the following lemma:

Lemma 7. *Let $\mathcal{A}$ be an automaton accepting $NF(R)$.There is a constant k_2 such that, if w is a transition sequence from a state q to itself, then there is a word u whose length is smaller than k_2 and an integer m satisfying $w = u^m$.*

Using this lemma together with standard combinatorial arguments [17], we get:

Lemma 8. *Let $\mathcal{A}$ be an automaton accepting $NF(R)$.There is a constant k_2 such that, if $w_1, \ldots, w_n$ are transition sequences, of length greater than k_2, from a state q to itself, then there is a word u whose length is smaller than k_2 and integers $m_1, \ldots, m_n$ satisfying $w_i = u^{m_i}$ for all i.*

Finally, thanks to arithmetical properties, we can flatten such automata:

Lemma 9. *$(u^{m_1} + \ldots + u^{m_n})^*$ is accepted by a flat automaton.*

Which, altogether, allows to prove theorem 1.

Example 9. Let us continue our running example. If we apply the algorithm sketched in this section and if we let $\alpha = ST(= abcd)$ and $\beta = T(= cd)$, then the rewrite system is

$$R = \{\alpha\beta x\alpha\alpha y\alpha \ \rightarrow \ \alpha\alpha y\alpha\beta x\alpha;\ \beta\beta x\beta\alpha y\beta \ \rightarrow \ \beta\alpha y\beta\beta x\beta \mid x, y \in (\alpha + \beta)^*\}$$

and the flat automaton (or rather its abstraction using the letters α, β instead of a, b, c, d: the whole flat automaton for $NF(R)$ would contain 76 states) is displayed on figure 5. All states are final.

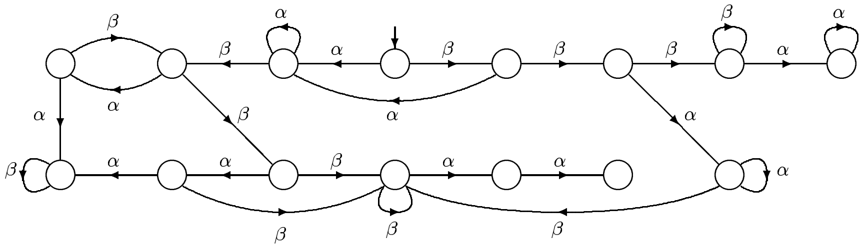

Fig. 5. The resulting flat automaton

3.4 The main theorem

Summing up what we have so far, thanks to lemma 3 and theorem 1, we can compute a flat automaton $\mathcal{A}_1$ which accepts at least one sequence of transitions for each equivalence class of $\approx$. We may moreover restrict the transition sequences accepted by $\mathcal{A}_1$ to transition sequences which are possible in the original automaton $\mathcal{A}$, thanks to a closure property for flat automata:

Lemma 10. *Let $\mathcal{A}$ be any finite automaton and $\mathcal{A}_1$ be a flat automaton. Then there is a flat automaton $\mathcal{A}'$ which accepts $L(\mathcal{A}) \cap L(\mathcal{A}_1)$.*

Moreover, there are two mappings f_1, f_2 from states of $\mathcal{A}'$ into respectively the states of $\mathcal{A}$ and the states of $\mathcal{A}_1$ such that, for every w and every states q, q', q_1, q'_1,

$$\left.\begin{array}{c} q \xrightarrow[\mathcal{A}]{w} q' \\ q_1 \xrightarrow[\mathcal{A}_1]{w} q'_1 \end{array}\right\} \Leftrightarrow \exists q_2, q'_2 . \left\{\begin{array}{l} q_2 \xrightarrow[\mathcal{A}']{w} q'_2 \\ f_1(q_2) = q, f_2(q_2) = q_1 \\ f_1(q'_2) = q', f_2(q'_2) = q'_1 \end{array}\right.$$

The proof of this lemma is similar to that of lemma 9: we construct $\mathcal{A}'$ from the product automaton $\mathcal{A} \times \mathcal{A}_1$, then all loops on a state are power of a same word, thanks to the flatness of $\mathcal{A}_1$. Now, we are able to prove our main theorem:

Theorem 2. *Every timed automaton can be emulated by a flat timed automaton.*

Proof. Let $\mathcal{A}$ be any timed automaton and let $\mathcal{A}_1$ be a flat automaton which accepts at least one transition sequence for each class modulo $\approx$. Such an automaton does exist thanks to lemmas 3, 4 and theorem 1 ($w \approx w'$ implies $\phi_w \models\!\mid \phi_{w'}$ as $\approx$ is the least congruence relation which satisfies the commutation properties of lemma 4). $\mathcal{A}$ can also be seen as a finite automaton on transition sequences and we construct $\mathcal{A}'$ as in lemma 10. $\mathcal{A}'$ can be seen as a timed automaton.

Now $\mathcal{A}'$ emulates $\mathcal{A}$: let f_1, f_2 be as in lemma 10. Then if $(q, \boldsymbol{V}) \xrightarrow[\mathcal{A}]{w} (q', \boldsymbol{V'})$, let $w' \approx w$. $(q, \boldsymbol{V}) \xrightarrow[\mathcal{A}]{w'} (q', \boldsymbol{V'})$, thanks to lemma 4. In particular if w' is the representative of the class of w w.r.t. $\approx$ which is accepted by $\mathcal{A}_1$ in state q'_1, then $q_1 \xrightarrow[\mathcal{A}_1]{w'} q'_1$ where q_1 is the initial state of $\mathcal{A}_1$. It follows, by lemma 10, that there are states q_2, q'_2 of $\mathcal{A}'$ such that $q_2 \xrightarrow[\mathcal{A}']{w'} q'_2$. Then $(q_2, \boldsymbol{V}) \xrightarrow[\mathcal{A}']{w'} (q'_2, \boldsymbol{V'})$. Conversely, if $(q_2, \boldsymbol{V}) \xrightarrow[\mathcal{A}']{w} (q'_2, \boldsymbol{V'})$, then $(f_1(q_2), \boldsymbol{V}) \xrightarrow[\mathcal{A}]{w} (f_1(q'_2), \boldsymbol{V'})$. Then it suffices to choose f_1 for the emulation function ϕ of definition 2.

Example 10. Considering the automaton of figure 1, our automatic construction will not yield the automaton of figure 2 (unfortunately). We actually obtain an automaton which is isomorphic to the automaton of figure 5.

4 Expressibility of the reachability relation

We use here a transformation of timed automata into automata with counters and use a result on flat automata with counters.

Definition 3 ([10]). *An* automaton with (real valued) counters *is a tuple* $(Q, q_i, C, \delta \subseteq Q \times G(C, C') \times Q)$ *where*

- Q *is a finite set of* states *and* $q_i \in Q$ *is an* initial state
- C *is a finite set of* counter names*;* C' *is the set of primed counter names.*
- $G(C, C')$ *is the set of* guards *built on the alphabets* C, C'*. A member of* $G(C, C')$ *is a conjunction of atomic formulas of one of the forms* $x \# y + c$*,* $x \# c$ *where* $x, y \in C \cup C'$*,* $\# \in \{\geq, \leq, =, >, <\}$ *and* $c \in \mathbb{R}$*.*

The automaton may *move* from a configuration $(q, \boldsymbol{v})$ to a configuration $(q', \boldsymbol{v'})$, which we write $(q, \boldsymbol{v}) \rightarrow (q', \boldsymbol{v'})$ if there is a triple $(q, g, q') \in \delta$ such that $\boldsymbol{v}, \boldsymbol{v'} \models g$, with the standard interpretation of relational symbols.

Example 11. Consider the automaton of figure 6.

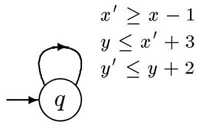

Fig. 6. A flat counter automaton

Possible moves are for instance: $(q, \binom{1}{1}) \rightarrow (q, \binom{0}{2})$ or $(q, \binom{1}{1}) \rightarrow (q, \binom{3}{3})$.

Following [13], timed automata can be seen as a particular class of automata with counters: we add a clock τ which is never reset and never used in the constraints (it measures the total elapsed time) and we use the variable transformation $x \mapsto \tau - x$. This yields a transformation on clocks valuations from $\boldsymbol{V}$ to $\boldsymbol{V_c}$. Then, if $< q, q', a, \phi, \lambda > \in E$, we translate it into a transition $\delta = < q, q', g >$ where g is the translation of ϕ together with the constraints $c' = \tau$ for each $c \in \lambda$ and $c' = c$ for each $c \notin \lambda$, plus the constraints on time positiveness: $\tau' \geq \tau$ and $c \leq \tau$ for every c. In this way each timed automaton $\mathcal{A}$ can be translated into an automaton with counters $\mathcal{A}_c$:

Theorem 3 ([13]). $(q, \boldsymbol{V}) \xrightarrow[\mathcal{A}]{*} (q', \boldsymbol{V'})$ *iff there is a vector* $\boldsymbol{V^1}$ *such that* $\exists t \geq 0$*,* $(q, \boldsymbol{V^1_c}) \xrightarrow[\mathcal{A}_c]{*} (q', \boldsymbol{V'_c})$ *and* $\boldsymbol{V^1} \models I(q), \boldsymbol{V} \models I(q), \boldsymbol{V^1} = \boldsymbol{V} + t\,\boldsymbol{1}$*.*

In other words, if we start a computation of $\mathcal{A}$ by firing a transition, then $\xrightarrow[\mathcal{A}]{*}$ is identical to $\xrightarrow[\mathcal{A}_c]{*}$, modulo the variable change.

On the other hand, we have the following result on flat automata:

Theorem 4 ([10]). *For every flat real (resp. integer) counter automata, the relations* $\xrightarrow[q,q']{*}$ *are effectively definable in* $\mathcal{T}$ *(resp. Presburger arithmetic).*

Now, from theorem 2, every timed automaton $\mathcal{A}$ can be emulated by a flat timed automaton $\mathcal{A}'$ and, by lemma 1, the reachability relations in $\mathcal{A}$ can be expressed as finite unions of the reachability relations of $\mathcal{A}'$. By theorem 4, the reachability relations of the flat automaton $\mathcal{A}'$ are expressible in the additive theory of real numbers. Hence it follows that:

Theorem 5. *For every timed automata, the binary reachability relations* $\xrightarrow[q,q']{*}$ *are effectively definable in the additive theory of real numbers.*

5 Examples of properties which can be decided on timed automata

Using the reachability relation, it is possible to express that some "good" (resp. "bad") thing may (resp. may not) happen. This is typical for safety properties. However it is not (at least in an obvious way) possible to express that something *must* happen. Typically, we cannot express inevitability. Hence, only a fragment of timed temporal logics can be expressed in the first-order theory of the reachability relation. However, our formalism offers other possibilities.

Example 12. On the simple automaton of figure 7, using the binary reachability relation, we can check properties of configurations, that are not properties of regions. This points out a difference in nature between our result and [18].

Consider the automaton of figure 7 wich contains only one clock x, and no constant. Assume that we are interested in the following property: " After firing a in configuration c, what is the minimal delay spent before reaching c again ?" For example, from state q_0 with $x = 0$ we can fire a and then b without waiting in q_1, then we are again in q_0 with $x = 0$. In this case the minimal delay is 0.

If the initial value x_0 of x is a parameter, then the minimal delay d is a function of x_0. In our example, we have $d(x_0) = x_0$. This result can be obtained using the binary reachability relation, since the set of possible delays is $\mathcal{D}(x_0) = \{\tau' - \tau \mid (q_0, x_0, \tau) \xrightarrow{ab} (q_0, x_0, \tau')\}$ and $d(x_0)$ is such that $d(x_0) \in \mathcal{D}(x_0)$ and $\forall d' \in \mathcal{D}(x_0), d(x_0) \leq d'$.

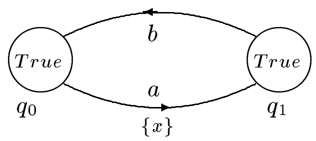

Fig. 7. A very simple timed automaton

Such a minimal delay property cannot be obtained using classical methods since, usually, the computed delays cannot depend on the initial configuration.

For instance, for the automaton of figure 7, there are two clock regions (depending on $x = 0$ or not). The minimal delay between two configurations of the region automaton is always 0.

More generally, it is possible to express sets of configurations which are not necessary unions of regions; any first-order definable set of clock values can be used in the logic. For instance we could express "each time the clock x is the double of clock y, we can reach a state in which y is a third of z, within a delay which is the half of x":

$$\forall \boldsymbol{X}, \exists \boldsymbol{X'}, (x = 2y) \Rightarrow (\boldsymbol{X} \xrightarrow[\mathcal{A}]{*} \boldsymbol{X'} \wedge z' = 3y' \wedge 2(\tau' - \tau) = x)$$

Not only such sets of configurations are expressible, but also relations between configurations. For instance we can express that "each time we are in state q, we can reach a configuration in which the clock x has doubled". This corresponds to a relation $x' = 2x$.

Using free variables, it is also possible to define values of clocks (or delays). For instance, the minimal delay between configurations c_1, c_2 can be defined by:

$$\phi(x) \stackrel{\text{def}}{=} \exists \tau, (c_1, \tau) \xrightarrow[\mathcal{A}]{*} (c_2, \tau + x) \wedge \forall y < x, \forall \tau, \neg(c_1, \tau) \xrightarrow[\mathcal{A}]{*} (c_2, \tau + x)$$

and hence can be computed or used in further verifications.

We can take advantage of the *binary* relation: it is as easy to express properties about the past as about the future. For instance: "each time we reach a state q, then the clock x was never larger than $y + z$ in the past".

We conclude this section with an example which looks more relevant: assume we have a server which receives requests from several users. Assume that the server receives requests from two users at time t and $t + \epsilon$ and that these requests are granted at time $t + \delta_1$ and $t + \delta_2$. We may want to check that the server is "fair" and that the delay δ_2 is always smaller than $2 \times \delta_1 + \epsilon$, for instance. This is again a typical property which can be expressed and checked in our theory.

6 Conclusion and perspectives

We have shown that, for timed automata, the binary reachability relation is definable in a decidable theory. The formula which results from our computation may be, in principle, quite huge since for instance the number of equivalence classes for $\sim$ is exponential. Hence our result is, so far, more of theoretical nature. One possible future research direction is then to have more precise informations on the complexity of the method (both theoretically and practically). However, beyond an hypothetical practical verification technique for timed automata, we believe that our result shows another possible direction of research. It may suggest other (more tractable) real-time computation models, starting from the logical side (the theory of real numbers). For instance, we could start from flat timed automata (without loosing expressiveness in the clock valuations sense). It also separates the expressiveness of the properties to be checked (in which it is possible to express much more relationships between clocks) from the expressiveness of the model. There are also several side effects of our proof. For instance we can decide whether or not a loop (or several loops) can be iterated.

The ability to express the reachability relation as a constraint between initial and final clock values allows to replace a whole automaton (or a piece of it corresponding to a timed automaton) with a single *meta-transition* [9], hence faithfully abstracting complex models. This can be used in verifying complex systems.

Conversely, we can mechanically check properties of models which are more expressive than timed automata.

Example 13. We consider two timed automata which are connected by a single transition whose enabling condition is an arbitrary first-order formula ψ over clock values (see figure 8). Properties of such a network can be verified as easily (or as hardly) as properties of a single timed automaton: it suffices to compute the binary reachability relation for each individual automaton and then to connect the two formulas with the enabling condition ψ.

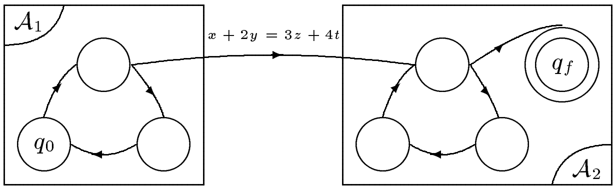

Fig. 8. An extended timed automaton

This can be extended of course to any network of timed automata, provided that there is no cycle through such general transitions.

Adding a stop watch to the model yields undecidability of the reachability [14]. However, as shown in [3], this does not imply that we cannot check properties involving accumulated delays. It is not possible, at least in an obvious way, to express accumulated delay constraints using the first-order theory of the reachability relation. We believe that it is still worth to study more deeply what can (and what cannot) be automatically checked using a similar approach.

Another interesting possible investigation consists in considering *parametrised* timed automata, as in [8]. Though the authors show that emptiness of such automata is undecidable as soon as there are at least three clocks, our method seems to be well-suited for parametric reasoning and, for instance, we may derive conditions on the control instead of on the number of clocks, which yield decision techniques for such parametrised automata.

Finally, our method seems to be well-suited for models which combine timed automata with additional global variables: assume we add registers to the model, with simple operations on them such as in definition 3, then, for flat timed automata with counters, we expect to get again a decision procedure.

References

1. R. Alur. Timed automata. In *Verification of Digital and Hybrid Systems, Proc. NATO-ASI Summer School, Antalya, Turkey*, 1997. To appear.
2. R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real time. *Information and Computation*, 104(1):2–24, 1993.
3. R. Alur, C. Courcoubetis, and T. Henzinger. Computing accumulated delays in real-time systems. In *Proc. 5th Conf. on Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 181–193. Springer-Verlag, 1993.
4. R. Alur and D. Dill. Automata for modeling real-time systems. In *Proc. 17th Int. Coll. on Automata, Languages and Programming, Warwick, LNCS 443*, pages 322–335. Springer-Verlag, 1990.
5. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
6. R. Alur, T. Feder, and T. Henzinger. The benefits of relaxing punctuallity. *J. ACM*, 43:116–146, 1996.
7. R. Alur and T. Henzinger. A really temporal logic. *J. ACM*, 41:181–204, 1994.
8. R. alur, T. Henzinger, and M. Vardi. Parametric real-time reasoning. In *Proc. 25th Annual ACM Symposium on Theory of Computing*, 1993.
9. B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Computer Aided Verification, Proc. 6th Int. Conerence*, LNCS, Stanford, June 1994. Springer-Verlag.
10. H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In A. Hu and M. Vardi, editors, *Proc. Computer Aided Verification*, volume 1427 of *LNCS*, pages 268–279, Vancouver, 1998. Springer-Verlag.
11. H. Comon and Y. Jurski. Timed automata and the theory of real numbers. Technical report, LSV Research Report, June 1999. avalaible at http://www.lsv.ens-cachan.fr/Publis/publis-lsv-index.html.
12. C. Courcoubetis and M. Yannakakis. Minimal and maximal delay problems in real time systems. In K. Larsen and A. Skou, editors, *Proc. CAV 91: Computer Aided Verification*, volume 575 of *Lecture Notes in Computer Science*, pages 399–409. Springer-Verlag, 1991.
13. L. Fribourg. A closed form evaluation for extending timed automata. Technical Report 1998-02, Laboratoire Spécification et Vérification, ENS Cachan, Mar. 1998.
14. T. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What is decidable about hybrid automata ? In *Proc. 27th Symposium on Theory of Computing*, pages 373–382. ACM Press, 1995.
15. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real time systems. *Information and Computation*, 111(2):193–244, 1994.
16. F. Laroussinie, K. Larsen, and C. Weise. From timed automata to logic – and back. In *Proc. 20th Conf. on Foundations of Computer Science*, volume 969 of *Lecture Notes in Computer Science*, Prag, 1995. Springer-Verlag.
17. M. Lothaire. *Combinatorics on words*, volume 17 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 1982.
18. F. Wang. Timing behaviour analysis for real time systems. In *Tenth Annual IEEE Symposium on Logic in Computer Science*, San Diego, CA, June 1995. IEEE Comp. Soc. Press.
19. T. Wilke and M. Dickhöfer. The automata-theoretic method works for tctl model checking. Technical Report 9811, Inst. f. Informatik u. Prakt. Math., CAU Kiel, 1998.