An Efficient Customized Clock Allocation Algorithm for a Class of Timed Automata

Neda Saeedloei[1] and Feliks Kluźniak[2]

[1] Towson University
nsaeedloei@towson.edu,
[2] RelationalAI
feliks.kluzniak@relational.ai

Abstract. We present a new clock allocation algorithm for a fairly general class of timed automata. The algorithm is customized to take advantage of the special properties of automata in this class, and is therefore efficient.

1 Introduction

Timed automata [**?**] have been used as a standard formalism for modelling real-time systems. A timed automaton is a finite automaton extended with a finite set of real-valued variables called clocks. Clocks appear in constraints that accompany transitions of the automaton and express conditions on the times at which the transitions can be taken: a transition r on symbol a can be taken if the input is a and the accompanying clock constraints are satisfied. When a transition is taken, it can reset some clocks, thus making it possible for constraints to refer to the time elapsed since the clocks were last reset. All the clocks of an automaton advance at the same rate.

Model checking [**?**] is a popular technique for formal verification of models of complex systems, including real-time systems. When applied to timed automata, it can be computationally expensive, and the cost crucially depends on the number of clocks [**?**]. Although, for a given timed automaton, it is in general undecidable whether there exists a language-equivalent automaton with fewer clocks [**?**], the problem of decreasing the number of clocks of timed automata has been an active area of research [**?**,**?**,**?**,**?**]. The existing approaches for tackling the problem are based on either the syntactic form (e.g., [**?**,**?**]) or the semantics (e.g., [**?**,**?**]) of timed automata. Regardless of the particular approach —syntax or semantics based—the problem has been addressed mostly by constructing bisimilar timed automata [**?**,**?**].

A more recent approach [**?**] is not based on constructing bisimilar timed automata, but on a compiler-like static flow analysis of a given automaton $\mathcal{A}$. This approach can be applied when $\mathcal{A}$ belongs to TA_S, the class of automata that have at most one clock reset on any transition, and moreover, every clock is well-defined, i.e., on any path from the initial location to the use of a clock in a constraint, the clock is reset before it is used. Given an automaton in TA_S, this

method finds the optimal (i.e., smallest) number of clocks for the equivalence class of all automata that have the same graph and the same pattern of clock resets and uses. That is, it is impossible to use a smaller number of clocks without violating the semantics of at least one member of the class (as long as all the members have their original graphs and constraints, modulo clock renaming).

This method performs a liveness analysis of clocks, the result of which is used to construct an interference graph. It is shown that a colouring of the interference graph with a minimum number of colours is equivalent to finding an optimal clock allocation for the original timed automaton, i.e., the optimal number of clocks is the chromatic number of the graph [**?**]. It is also shown that for every automaton in $TA_{DS} \subsetneq TA_S$ (see Sec. 3) the interference graph is chordal (therefore perfect), hence it can be coloured in polynomial time [**?**], once the interference graph is constructed whose worst-case complexity is $O(n^3)$.

The class TA_{DS} was first encountered in the context of synthesizing timed automata from a set of timed scenarios [**?**]. This is the class of automata that satisfy two properties [**?**]: (i) a given clock is reset on all the outgoing transitions of a unique location, and (ii) if a clock constraint $t_j \sim a$, for some clock t_j and some $a \in \mathbb{Q}$, appears on an outgoing transition of location s, then clock t_j must be reset on an outgoing transition of a location q, where q dominates s (i.e., all paths from the initial location to s pass through q [**?**]).

The current paper considers the class TA_{DS} and develops a new clock allocation method which is customized for this class. Our method is based on a new liveness analysis algorithm, which is much simpler and more efficient, as it takes advantage of the properties of automata in TA_{DS}. These properties allow us to formulate a clock allocation algorithm that is also efficient. Given an automaton $\mathcal{A}$ in TA_{DS}, the algorithm computes an optimal ("optimal" in the sense used in the cited work [**?**]) clock allocation for $\mathcal{A}$ and replaces the original clocks with a new set of clocks, without changing the graph or the language of $\mathcal{A}$.

2 Timed automata

A *timed automaton* [**?**] is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, Q_f, C, T \rangle$, where Σ is a finite alphabet, Q is the (*finite*) set of locations, $q_0 \in Q$ is the initial location, $Q_f \subseteq Q$ is the set of final locations, C is a finite set of *clock* variables (clocks for short), and $T \subseteq Q \times Q \times \Sigma \times 2^C \times 2^{\Phi(C)}$ is the set of transitions. In each transition $(q, q', e, \lambda, \phi)$, λ is the set of clocks to be reset with the transition and $\phi \subset \Phi(C)$ is a set of clock constraints over C of the form $c \sim a$ (where $\sim \in \{\leq, <, \geq, >, =\}$, $c \in C$ and a is a constant in the set of rational numbers, $\mathbb{Q}$).

A *clock valuation*, ν, for C is a mapping from C to $\mathbb{R}^{\geq 0}$. ν, *satisfies* a set of clock constraints ϕ over C iff every clock constraint in ϕ evaluates to true after each clock c is replaced with $\nu(c)$. For $\tau \in \mathbb{R}, \nu + \tau$ denotes the clock valuation which maps every clock c to the value $\nu(c) + \tau$. For $Y \subseteq C$, $[Y \mapsto \tau]\nu$ is the valuation which assigns τ to each $c \in Y$ and agrees with ν over the rest of the clocks.

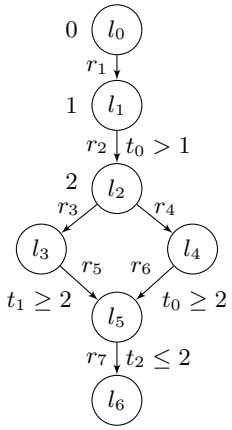

Fig. 1. In TA_{DS}

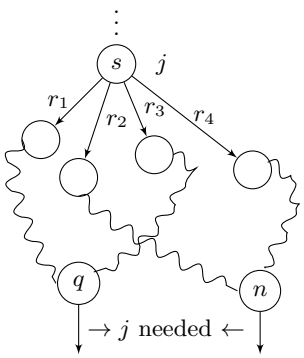

Fig. 2. Problematic locations

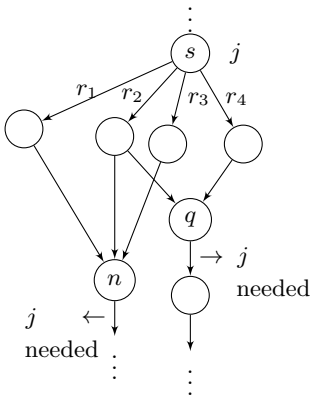

Fig. 3. Families

A *timed word* over an alphabet Σ is a pair (σ, τ) where $\sigma = \sigma_1\sigma_2...$ is a finite [**?**, **?**] or infinite [**?**] word over Σ and $\tau = \tau_1\tau_2...$ is a finite or infinite sequence of (time) values such that (i) $\tau_i \in \mathbb{R}^{\geq 0}$, (ii) $\tau_i \leq \tau_{i+1}$ for all $i \geq 1$, and (iii) if the word is infinite, then for every $t \in \mathbb{R}^{\geq 0}$ there is some $i \geq 1$ such that $\tau_i > t$. A run ρ of $\mathcal{A}$ over a timed word (σ, τ) is a sequence of the form $\langle q_0, \nu_0\rangle \xrightarrow[\tau_1]{\sigma_1} \langle q_1, \nu_1\rangle \xrightarrow[\tau_2]{\sigma_2} \langle q_2, \nu_2\rangle \xrightarrow[\tau_3]{\sigma_3} \ldots$, where for all $i \geq 0$, $q_i \in Q$ and ν_i is a clock valuation such that (i) $\nu_0(c) = 0$ for all clocks $c \in C$ and (ii) for every $i > 1$ there is a transition in T of the form $(q_{i-1}, q_i, \sigma_i, \lambda_i, \phi_i)$, such that $(\nu_{i-1} + \tau_i - \tau_{i-1})$ satisfies ϕ_i, and ν_i equals $[\lambda_i \mapsto 0](\nu_{i-1} + \tau_i - \tau_{i-1})$. The set $inf(\rho)$ consists of $q \in Q$ such that $q = q_i$ for infinitely many $i \geq 0$ in the run ρ.

A run over a finite timed word is *accepting* if it ends in a final location [**?**]. A run ρ over an infinite timed word is *accepting* iff $inf(\rho) \cap Q_f \neq \emptyset$ [**?**]. The *language* of $\mathcal{A}$, $L(\mathcal{A})$, is the set $\{(\sigma, \tau) \mid \mathcal{A}$ has an accepting run over $(\sigma, \tau)\}$.

3 The Class TA_{DS}

Most of the definitions in this section and Sec. 5.1 are taken from our earlier work [**?**].

Assume $\mathcal{A} = \langle E, Q, \{q^0\}, Q_f, V, R\rangle \in TA_{DS}$ is the original timed automaton. If $r = (q, q', e, \lambda, \phi) \in R$, then $source(r) = q$ and $target(r) = q'$. The outgoing/incoming transitions of a location $s \in Q$ are denoted by $out(s) = \{r \in R \mid s = source(r)\}$ and $in(s) = \{r \in R \mid s = target(r)\}$.

Let $\mathcal{A}$ be a timed automaton with a unique initial location. If s and q are locations in $\mathcal{A}$, then s *dominates* q if and only if all paths from the initial location to q pass through s [**?**]. We follow the conventions of the cited work [**?**] and denote the *dominance relation* between locations in $\mathcal{A}$ by $\succeq$: $s \succeq q$ iff s dominates q (we also say that q is dominated by s). We write $s \succ q$ to denote that $s \succeq q$

and $s \neq q$. The definition of dominated locations can be extended to *dominated transitions*: a transition r is *dominated* by location s iff $s \succeq source(r)$.

In the timed automaton of Fig. 1, l_0, l_1, l_2, l_5 and l_6 are all the locations that dominate l_6. Transition r_7 is dominated by l_0, l_1, l_2 and l_5.

In the rest of the paper we assume that every timed automaton $\mathcal{A}$ has an associated injective partial labelling function $L : Q \to N_L$, where $N_L \subset \{0, 1, 2, ...\}$ is the set of labels (values of L) used for $\mathcal{A}$.

Definition 1 *A timed automaton belongs to the class TA_{DS} if and only if it satisfies the following three restrictions:*

1. *It has a unique initial location, q^0, from which every location can be reached.*
2. *Clock t_j is reset on all the transitions in $out(s)$, where $L(s) = j$.*[1]
3. *A clock constraint on a transition r can contain an occurrence of t_j* only *if $j \in N_L$ and $L^{-1}(j) \succ source(r)$.*

Restriction 3 is called *the dominance assumption*, and it means that if $t_j \sim a$ is a clock constraint on a transition $r \in out(s)$, then the value of t_j represents the amount of time that has elapsed since leaving a location q, where $q \succ s$ and $L(q) = j$. This guarantees that all clocks are *well-defined*: a reference to t_j in a clock constraint on r is always preceded by a reset, on every path from q^0 to r.

4 The notion of optimality

To make the paper self-contained, in this section we describe the notion of optimality used in our earlier work [**?**].

Assume $\mathcal{A} = \langle E, Q, Q_0, Q_f, V, R\rangle$ is the original timed automaton, where $Q_f \neq \emptyset$.[2] We assume that the clocks in $\mathcal{A}$ belong to the set $V = \{t_0, t_1, t_2, ...\}$. Our clock allocation method will replace these with a disjoint set of clocks.

Let $N = \{j \mid t_j \sim a \in \phi_r$, where $r \in R\}$. This is the set of *clock numbers*, i.e., of the indices of all the clocks that are referred to on some transitions in R. Notice that $N \subseteq N_L$.

For a transition $r = (s, q, e, \lambda, \phi) \in R$, we use ϕ_r to denote the set of clock constraints on r.

Definition 2 *$clock\_ref : R \to 2^N$ maps transition r to the set $\{j \mid t_j \sim a \in \phi_r\}$. Intuitively, $clock\_ref(r)$ is the set of (indices of) clocks that are referred to in the constraints on r.*

Given a timed automaton $\mathcal{A}$, we abstract from the specific syntax of the constraints and consider only the identities of the clocks that are used in them. This concept is formalized as follows:

[1] If a transition leads only to paths on which t_j is not used, the reset (and the clock that is reset) will be eliminated by our algorithms.

[2] One could argue that the optimal number of clocks of a timed automaton is zero when the accepted language is empty.

Definition 3 *Let* $\mathcal{A} = \langle E, Q, Q_0, Q_f, V, R\rangle$*. The syntactic abstraction of* $\mathcal{A}$ *is defined as* $AbsSynt(\mathcal{A}) = \langle E, Q, Q_0, Q_f, V, R_c\rangle$*, where*
$R_c = \{(s, q, e, \lambda, clock\_ref(r)) \mid r = (s, q, e, \lambda, \phi) \in R\}$.

Let *TA* be a given set of timed automata. The function *AbsSynt* induces a "natural" division of *TA* into equivalence classes[3] (of course, this has nothing to do with the equivalence of automata in the usual sense). We will use $C(\mathcal{A})$ to denote the equivalence class of $\mathcal{A} \in TA$, i.e., $C(\mathcal{A}) = \{a \in TA \mid AbsSynt(a) = AbsSynt(\mathcal{A})\}$.

Given a timed automaton $\mathcal{A} \in TA$, our method of clock allocation is carried out on $AbsSynt(\mathcal{A})$. It will be clear from the construction that the clock allocation is correct and optimal for the entire equivalence class of $\mathcal{A}$, in the following sense:

1. *Correctness:* if we systematically replace the clocks (as prescribed by the computed allocation) in any timed automaton belonging to $C(\mathcal{A})$, then the resulting automaton will accept the same language as the original one.
2. *Optimality:* any allocation with fewer clocks would be incorrect, i.e., there would exist at least one timed automaton $\mathcal{B} \in C(\mathcal{A})$, for which it would change the accepted language.

The reason we perform this abstraction is that in our analysis we do not want to concern ourselves with "pathological" cases in which the required number of clocks is strongly affected by some particular form of the constraints. An example of such a pathological case is an automaton with a clock c, such that all the clock constraints on c are of the form $c \geq 0$, which is always true. Clearly, in an optimal clock allocation clock c can be safely removed.

5 Finding an Optimal Allocation of Clocks

Our objective is to transform a timed automaton $\mathcal{A} \in TA_{DS}$ to an equivalent automaton $\mathcal{A}'$ with the same graph and the same set of constraints (modulo clock renaming), but with a number of clocks that is *optimal* in the sense defined in Sec. 4. This is done in four steps:

1. *Liveness analysis* identifies the *ranges* of clocks in $\mathcal{A}$.
2. The original clocks are replaced with new clocks in a way that takes ranges into account but minimizes the number of clocks.
3. Old clock resets are deleted and new clock resets are generated.
4. Existing clock constraints are rewritten in terms of the new clocks.

5.1 Liveness Analysis of Clocks

Assume $\mathcal{A} = \langle E, Q, \{q^0\}, Q_f, V, R\rangle \in TA_{DS}$ is the original timed automaton and L is the aforementioned labelling function.

[3] Let $f : X \to Y$ be a total function. For any $D \subseteq X$, $\{(a, b) \in D^2 \mid f(a) = f(b)\}$ is an equivalence relation.

If $r = (s, q, e, \lambda, \phi) \in R$, then λ_r is used to denote the set of clocks to be reset on r. Notice that $|\lambda_r| \leq 1$ (since $\mathcal{A} \in TA_{DS}$).

If $p = r_1...r_k$ is a path, then $transitions(p) = \{r_1, ..., r_k\}$.

We use the following functions:

- $active : R \to 2^N$ maps transition r to the set $\{j \mid$ there is a path $rr_1...r_k$, $k \geq 1$, such that $j \in clock\_ref(r_k)$ and $t_j \notin \lambda_{r_i}$ for $1 \leq i < k\}$. Intuitively, $active(r)$ identifies clocks that are "alive" on r (i.e., their values may be subsequently used).
- $born : R \to 2^N$ maps transition r to the set $\{j \mid t_j \in \lambda_r \wedge j \in active(r)\}$. Intuitively, $born$(r) identifies a clock that is reset on r whose value can be used on some transition reachable from r. Notice that $born(r) \subseteq active(r)$.
- $last\_ref : R \to 2^N$ maps transition r to $clock\_ref(r) \setminus active(r)$.
- $needed : R \to 2^N$ maps transition r to $active(r) \cup last\_ref(r)$.

A clock t_j *is needed* (or *is active*) on a transition r if $j \in needed(r)$ (or $j \in active(r)$).

Let r be a transition such that $j \in last\_ref(r)$. Then in the target timed automaton a (new) clock c that has been assigned to the old clock t_j can be reassigned to another (old) clock and be reset on r.

In the automaton of Fig. 1, 0 is in both $active(r_2)$ and $clock\_ref(r_2)$, but not in $last\_ref(r_2)$. On r_6, $0 \in clock\_ref(r_6)$, but $0 \notin active(r_6)$, so $0 \in last\_ref(r_6)$.

Lemma 1 *For a timed automaton in TA_{DS} with its set of locations Q, we have* $\forall_{q \in Q} \forall_{r_i, r_k \in in(q)}\ active(r_i) = active(r_k)$.

Proof. Assume $j \in active(r_i)$. Then there is a transition r in $out(q)$ such that $j \in needed(r)$. But r can be reached from r_k, therefore $j \in active(r_k)$. The rest of the proof follows from symmetry. □

Definition 4 *A path $p = r_0...r_n$ is a* path for clock t_j *iff $born(r_0) = \{j\}$ and $j \in needed(r_i)$ for $0 \leq i \leq n$.*

In the automaton of Fig. 1, $r_1r_2r_4r_6$ is a path for clock t_0, as is $r_1r_2r_4$, r_1r_2 or just r_1. Similarly, $r_2r_3r_5$ is a path for clock t_1. Finally, $r_3r_5r_7$ and $r_4r_6r_7$ are paths for clock t_2.

Definition 5 *$range : N \times R \to 2^R$ maps (j, r) to $\{r' \mid r' \in transitions(p)$, where p is a path for clock t_j that starts at $r\}$.*

Intuitively, $range(j, r)$, where $j \in born(r)$, is the set of transitions in all the paths for clock t_j that begin at r. If $range(j, r) \neq \emptyset$, then we say it is a *range for clock t_j*.

In the automaton of Fig. 1, $range(0, r_1) = \{r_1, r_2, r_4, r_6\}$, $range(1, r_2) = \{r_2, r_3, r_5\}$, $range(2, r_3) = \{r_3, r_5, r_7\}$, and $range(2, r_4) = \{r_4, r_6, r_7\}$.

We will use the term *active range* to denote that set of transitions in the range on which the clock in question is active.

Liveness analysis of clocks is performed by Algorithm 1. Given an automaton $\mathcal{A} \in TA_{DS}$, the algorithm determines, in effect, the ranges of all the clocks in $\mathcal{A}$.

We assume the existence of the following four functions:

Algorithm 1: Computing *active* and *born*

Input : A timed automaton $\mathcal{A} = \langle E, Q, \{q^0\}, Q_f, V, R\rangle$, a labelling function L, functions *ancestor_transitions* and *dominated*.
Output: Functions *active* and *born*.

```
foreach transition r ∈ R in A do
    born(r) := active(r) := ∅;
foreach clock number j ∈ N do
    active_on(j) := ∅;
foreach transition r ∈ R in A do
    foreach clock number j ∈ clock_ref(r) do
        /* Dealing with cycles.                                   */
        tmp := ancestor_transitions(r) \ trans_to_source(r);
        tmp := tmp ∩ dominated(L^{-1}(j));
        if tmp ∩ out(target(r)) ≠ ∅ then
            tmp := tmp ∪ {r};
        active_on(j) := active_on(j) ∪ tmp;
foreach transition r = (s, q, e, λ, φ) ∈ R of A do
    if L(s) = j and r ∈ active_on(j) then
        born(r) := {j};
foreach clock number j ∈ N do
    foreach transition r ∈ active_on(j) do
        active(r) := active(r) ∪ {j};
```

- *ancestor_transitions* : $R \to 2^R$ maps transition r to the set of transitions from which it can be reached by some non-empty path.
- *dominated* : $Q \to 2^R$ maps location q to the set of transitions that are dominated by q.
- *active_on* : $N \to 2^R$ maps clock number j to the set of transitions on which j is active.
- *trans_to_source* : $R \times N \to 2^R$ maps transition r and clock number j to the set of transitions on all paths that begin with r and end at $L^{-1}(j)$.

Assuming that sets are implemented as bit vectors and that bit vector operations take constant time, the worst-case complexity of Algorithm 1 is $O(|R||N|)$.

For the automaton of Fig. 1, we show the values of *active* and *born* obtained by Algorithm 1 for two transitions: $born(r_4) = \{2\}$, $active(r_4) = \{0, 2\}$ and $born(r_6) = \emptyset$, $active(r_6) = \{2\}$. Notice that both clocks t_0 and t_2 are needed at r_4 and r_6.

5.2 Clock Allocation

After liveness analysis has generated the annotations in our automaton, the next step is to use these annotations to allocate new clocks. This is done by an algorithm that replaces the clocks of the original timed automaton with new

ones, while minimizing their number. The general idea is that, once a clock is reset on a transition on which it is born, it should never be reset again within the relevant active range; however, the clock can be reused outside that active range, provided that certain consistency requirements are satisfied.

We begin by introducing some additional notation.

Let A, B and C be sets and let $r \subset A \times B \times C$. The relation r can be applied to an argument $a \in A$ by using the left-associative operator ".": $r.a = \{(b, c) \mid (a, b, c) \in r\}$. Similarly, for $b \in B$, $r.a.b = \{c \mid (b, c) \in r.a\}$.

If, for every $(a, b) \in A \times B$, $r.a.b$ is either a singleton or the empty set, then r is a function of two arguments: $r : A \times B \to C$. If $r.a.b$ is never the empty set, then the function is total, otherwise it is partial.

If r is a function of two arguments, then $r.a.b = \{c\}$ is often written as $r(a, b) = c$ and $r.a.b = \{\}$ as $r(a, b) = \bot$.

Next, we formally define clock allocations. We assume the existence of a set $\mathcal{C}$, disjoint from V, with $|R|$ clock variables (this is enough, since at most one clock is reset on any given transition).

Definition 6 *Given a timed automaton $\mathcal{A}$ with the set R of transitions and the set N of clock numbers, a* clock allocation *for $\mathcal{A}$ is a relation $alloc \subset R \times \mathcal{C} \times N$ such that $(r, c, j) \in alloc \Rightarrow j \in active(r)$.*

Inclusion of (r, c, j) in *alloc* represents the fact that on transition r clock c is associated with the old clock t_j (i.e., c will eventually replace t_j on r).

Definition 7 *A clock allocation alloc is* inconsistent *iff there exist two overlapping paths for some clock t_j, p and p' (which need not be different), some $c \in \mathcal{C}$ and $r_k, r_l \in transitions(p) \cup transitions(p')$ such that*

$j \in active(r_k) \wedge j \in active(r_l) \wedge (r_k, c, j) \in alloc \wedge (r_l, c, j) \notin alloc$.

alloc is consistent *iff it is not inconsistent.*

(Notice that the paths may overlap on a transition r on which the clock is not active, i.e., we may have $t_j \in last\_ref(r)$.)

Definition 8 *A clock allocation alloc is* correct *if:*

- *alloc is a function, i.e., $alloc : R \times \mathcal{C} \to N$;*
- *alloc is consistent.*

Intuitively, the fact that *alloc* is a function means simply that, on any given transition, a clock c can be associated with at most one (old) clock t_j. Note that it is possible for a correct allocation to associate two or more (new) clocks with the same (old) clock on the same path. It is also possible for different clocks to be associated with the same (old) clock t_j on different paths for t_j, as long as the paths are disjoint.

Definition 9 *A clock allocation is* lean *if it is an injective function.*

Intuitively, a lean allocation does not associate a clock on a transition with more than one (new) clock.

Definition 10 *A clock allocation alloc is* complete *iff, for every transition* $r \in R$ *and every* $j \in active(r)$*, there is a clock* $c \in \mathcal{C}$ *such that* $(r, c, j) \in alloc$*.*

Observation 1 *Let* $\mathcal{A}$ *be a timed automaton with the set of transitions* R*, and let alloc be a complete, correct and lean clock allocation for* $\mathcal{A}$*. Then the following holds:* $\forall_{r \in R}\ |alloc.r| = |active(r)|$.

Definition 11 *We define the number of clocks used in an allocation by:*
$cost(alloc) = |\{c \in \mathcal{C} \mid \exists_{r \in R} \exists_{j \in N}\ (r, c, j) \in alloc\}|$.

Definition 12 *Let* $\mathcal{A}$ *be a timed automaton and let alloc be a complete and correct clock allocation for* $\mathcal{A}$*. The allocation alloc is* optimal *if there is no complete correct allocation* $alloc'$ *for* $\mathcal{A}$ *such that* $cost(alloc') < cost(alloc)$ *. [?]*

Theorem 1 *Let* $\mathcal{A}$ *be a timed automaton with the set of locations* Q*, and let alloc be a complete and correct clock allocation for* $\mathcal{A}$*. Then the following holds:*
$\forall_{s \in Q} \forall_{r_i, r_k \in in(s)}\ alloc.r_i = alloc.r_k$.

Proof. $alloc.r_i = alloc.r_k$ is equivalent to
$\forall_{j \in N} \forall_{c \in \mathcal{C}} ((r_i, c, j) \in alloc \Leftrightarrow (r_k, c, j) \in alloc)$.
Let $s \in Q$ and $r_i, r_k \in in(s)$. Let j and c be such that $(r_i, c, j) \in alloc$. This implies that $j \in active(r_i)$. But then, from Lemma 1, we have $j \in active(r_k)$. Therefore there is some $r \in out(s)$ such that both the sequences $r_i r$ and $r_k r$ belong to paths for clock t_j (because j is needed on r). Since the paths overlap on r, from consistency of *alloc* we must have $(r_k, c, j) \in alloc$. The rest of the proof follows from symmetry. □

In the timed automaton of Fig. 1, assume that c_0 is associated with clock t_0, c_1 with clock t_1 and c_2 with clock t_2. Then,
$\alpha = \{\ (r_1, c_0, 0), (r_2, c_0, 0), (r_2, c_1, 1), (r_3, c_1, 1), (r_3, c_2, 2),$
$(r_4, c_0, 0), (r_4, c_2, 2), (r_5, c_2, 2), (r_6, c_2, 2)\ \}$
is a lean, correct and consistent allocation. Notice that $1 \in last\_ref(r_5)$, so $active(r_5)$ does not include clock number 1. Similarly, clock t_0 is not active on transition r_6. So $\alpha.r_5 = \alpha.r_6 = \{(c_2, 2)\}$, in accordance with Theorem 1.

The theorem has two important implications for an algorithm that computes a clock allocation:

- Information about the allocations on all the incoming transitions of a location can be stored in the location itself.
- The exact order in which the transitions of the automaton are traversed need not be important.

These points will become clear as we present our clock allocation algorithm.

5.3 The Clock Allocation Algorithm

We describe our algorithm in two steps: in the first step we present the algorithm for tree-shaped automata, where the root of the tree is the initial location; in the second step we extend the algorithm to allocate clocks for an arbitrary timed automaton in TA_{DS}. We do so not because we think that tree-shaped automata are particularly important, but in the hope of helping the reader's intuition.

The Clock Allocation Algorithm for Tree-shaped Automata The tree variant is presented as Algorithm 2, which begins with an initial pool of available clocks, $\mathcal{C}$, and the set of used clocks, $\mathcal{U}$, which is initially empty. We assume that the clocks in $\mathcal{C}$ are numbered: the algorithm always allocates that available clock which has the smallest number.

The algorithm performs a depth first walk of the automaton, beginning at the initial location, and annotates each location with a set of available clocks and a set of *clock assignments*. An assignment is a pair (c, j), where c is the clock that replaces the (old) clock t_j. More precisely, we define the following functions:

- $pool : Q \to 2^{\mathcal{C}}$ maps a location s to the set of clocks available at s;
- $assignments : Q \to 2^{\mathcal{C} \times N}$ maps s to the set of clock assignments at s.

As the algorithm annotates each location s with $assignments(s)$ and $pool(s)$, it ensures that every (old) clock t_j and every (new) clock c appear at most once in $assignments(s)$, and a clock c appears either in $assignments(s)$ or in $pool(s)$.

Every time the algorithm visits a transition where a clock, e.g., t_j is born, it associates a clock, say c, with j. If c has never been used before, it is added to $\mathcal{U}$. When the algorithm visits a transition r where $j \in last\_ref(r)$, it restores c to the pool of available clocks. When the algorithm stops, $\mathcal{U}$ contains the clocks of the target automaton: $|\mathcal{U}|$ can be significantly smaller than $|\mathcal{C}|$.

Observe that the resulting allocation is obviously complete. Also, the following theorem holds:

Theorem 2 *The allocation found by Algorithm 2 is correct and lean.*

Proof. On any transition where a clock, say t_j, of the original timed automaton is born, one (new) clock, e.g., c, is assigned to j. Moreover, c is not, on that transition, assigned to any other i (corresponding to clock t_i). A clock is assigned to a clock number i only when t_i is born. So the allocation is always an injective function, i.e, it is lean.

To see that the allocation is consistent, take any path p for some clock t_j. A clock c is assigned to j only on the first transition of p. At that point c is removed from the pool and is not returned there before p ends. It is therefore impossible for c to be assigned to any i on any transition r of p such that $j \in active(r)$.

Observe that if there is another path for t_j, say p', that shares a transition with p, then p and p' will have a common initial transition, because the graph of the automaton is a tree. In that case c will be assigned to j on that transition, and will not be assigned to another clock number on any transition r of p or p' such that $j \in active(r)$. □

We now turn our attention to the general case, i.e., when the timed automaton $\mathcal{A}$ is not necessarily a tree.

The Clock Allocation Algorithm for General Automata Algorithm 2, when applied to general automata, would trace out a spanning tree of $\mathcal{A}$, and the resulting allocation would still be an injective function. It might not, however, be consistent. An inconsistency can arise only in a situation similar to the one

Algorithm 2: Assigning clocks to a tree-shaped automaton in TA_{DS}

Input : A timed automaton $\mathcal{A} = \langle E, Q, \{q^0\}, Q_f, V, R\rangle$, the initial pool of available clocks $\mathcal{C}$, and functions *active* and *born*.
Output: Functions *pool* and *assignments*, and the set $\mathcal{U} \subseteq \mathcal{C}$ of clocks used.

$\mathcal{U} := assignments(q^0) := \emptyset$;
$pool(q^0) := \mathcal{C}$;
foreach $r = (s, q, e, \lambda, \phi) \in R$, *in a depth first order* **do**
  if q *has not been visited yet* **then**
    $tmp\_pool := pool(s)$;
    $tmp\_assignments := assignments(s)$;
    $source\_active := \{j \mid (c, j) \in assignments(s)$, for some $c\}$;
    foreach $j \in source\_active \setminus active(r)$ **do**
      `/* j ∈ last_ref(r) */`
      $tmp\_assignments := tmp\_assignments \setminus \{(c, j)\}$, where $(c, j) \in assignments(s)$;
      $tmp\_pool := tmp\_pool \cup \{c\}$, where $(c, j) \in assignments(s)$;
    if $born(r) \neq \emptyset$ **then**
      $tmp\_pool := tmp\_pool \setminus \{d\}$, where d is the clock in tmp_pool which has the smallest number;
      $tmp\_assignments := tmp\_assignments \cup \{(d, j)\}$, where $born(r) = \{j\}$;
      $\mathcal{U} := \mathcal{U} \cup \{d\}$;
    $pool(q) := tmp\_pool$;
    $assignments(q) := tmp\_assignments$;

illustrated in Fig. 2. Two paths for a clock t_j have different initial transitions: r_1 and r_3. If on each of those transitions j were associated with a different (new) clock, then the values of the allocation on the paths that join at location q, in particular, on the transitions belonging to $in(q)$ would be different. So, Theorem 1 would not hold, even though the allocation would be complete and lean. We will call location q a *problematic location with respect to* j. Note that j is needed in the outgoing transition of q. The algorithm must be augmented to ensure that the same clock is assigned to j on all transitions in $out(s)$ (where $L(s) = j$) that can lead to the same problematic location q.

These considerations can be formalised as follows:

Definition 13 *Let $R_0 = range(j, r_0)$ and $R_1 = range(j, r_1)$ be two ranges of a clock t_j that start on some transitions r_0 and r_1. R_0 is* related *to R_1 iff (1) R_0 and R_1 intersect, or (2) R_1 intersects with some range of clock t_j, say R_2, and R_2 is related to R_0.*

Definition 14 *Let S be a range of clock t_j. We use $Rel(j, S)$ to denote the set of all ranges of t_j that are related to S.*

Notice that a range is related to itself, and $Rel(j, S)$ can be a singleton. Moreover, the set of ranges of clock t_j is partitioned by sets of the form $Rel(j, p)$, where p is a range for clock t_j.

Definition 15 *Let $A = Rel(j, S)$ for some j and S. The set of all those transitions belonging to members of A on which t_j is active is a* family *for t_j.*

In Fig. 1, $range(2, r_3) = \{r_3, r_5, r_7\}$ and $range(2, r_4) = \{r_4, r_6, r_7\}$ are related. If we choose, say, $S = range(2, r_3)$, then $Rel(2, S) = \{\{r_3, r_5, r_7\}, \{r_4, r_6, r_7\}\}$. The only family for t_2 is $\{r_3, r_4, r_5, r_6\}$ (t_2 is not active on r_7).

Observe that each family for a clock t_j must begin at the same location: location s, where $L(s) = j$. It is easy to determine whether two initial transitions for some clock t_j belong to the same family: we must check whether they can lead to the same problematic location.

In the automaton of Fig. 2 we see two families for clock t_j. The first is generated by paths for t_j that begin with r_1 and r_3: those transitions in these paths on which t_j is active form a family, because of the problematic location q. The second family is generated by paths that begin with r_2 and r_4.

Fig. 3 illustrates the transitive nature of a family: r_1, r_2 and r_3 belong to the same family, because of the problematic location n. Similarly, r_2 and r_4 belong to the same family on account of the problematic location q. Therefore r_1, r_2, r_3 and r_4 all belong to the same family.

Observation 2 *Let F be a family for some clock t_j, and let alloc be a complete correct allocation. Then there must exist a clock variable $c \in \mathcal{C}$ such that $alloc.r.c = \{j\}$ for every transition $r \in F$ on which t_j is active. Otherwise alloc would be inconsistent. We say that c is* allocated *to F.*

In the automaton of Fig. 2, t_j must be associated with the same (new) clock, say c, on paths for clock t_j that begin at transitions r_1 and r_3. Similarly, t_j must be associated with the same clock (which can, but need not, be c) on paths that begin at r_2 and r_4, as their transitions belong to the same family.

Observation 3 *The number of families to which a transition r belongs is $|active(r)|$.*

Proof. Assume the number of families to which transition r belongs is n. Therefore, there are exactly n families for different clocks that share transition r (two families for the same clock cannot share r, or they would be the same family). Since t_j is active on all the transitions of a family for t_j, it follows that $active(r)$ contains exactly one element for each of the n families. □

Definition 16 *Two families, F_1 and F_2, belong to the same* cluster *iff $F_1 \cap F_2 \neq \emptyset$. A cluster cl is a maximal set of such families, i.e., every family outside cl does not overlap with at least one of the families in cl.*

Observe that the members of a cluster must be families for different clocks (otherwise they would not overlap). Observe also that a family can belong to more than one cluster.

We say transition r *belongs* to cluster $\mathcal{F}$, if there is a family $F \in \mathcal{F}$, such that $r \in F$. Notice that if $active(r) = \emptyset$, then r does not belong to any family; therefore, it does not belong to any cluster.

Since each pair of families in a cluster shares some common transition, we immediately see the following:

Observation 4 *Every family in a cluster must be allocated a different clock.*

In the automaton of Fig. 1, there is only one cluster, and it contains all the three families: $\{r_1, r_2, r_4\}$ for clock t_0, $\{r_2, r_3\}$ for clock t_1 and $\{r_3, r_4, r_5, r_6\}$ for clock t_2. Obviously, a correct allocation for the automaton requires three clocks, even though no more than two clocks are needed on any particular transition.

Definition 17 *The* size *of a cluster is the cardinality of the set of families that form the cluster.*

Theorem 3 *Let alloc be a complete correct allocation for a timed automaton $\mathcal{A}$. Then cost(alloc) cannot be smaller than the size of the largest cluster in $\mathcal{A}$.*

Proof. This is a direct consequence of Observations 2 and 4. □

We are now ready to return to a detailed description of the general algorithm.

To take into account the problematic locations, at each location s labelled with some j, the outgoing transitions of s are divided into two sets: (i) the set of "mother" transitions, i.e., $\{r \in out(s) \mid born(r) = \{j\}\}$ and (ii) the set of "other" transitions, i.e., $\{r \in out(s) \mid born(r) = \emptyset\}$. Observe that the mother transitions are the initial transitions of all the families for clock t_j. The mother transitions require special attention, while the other transitions are processed as before. The set of mother transitions is divided into groups. Each group is the set of initial transitions of a family for clock t_j. *All transitions belonging to the same group must obtain the same clock assignment for clock number j* (see Observation 2). Notice that if for some clock, say t_i, $i \in last\_ref(r)$ for *every* transition r in the group, then the clock previously assigned to i becomes available and can be assigned to j.

It is a direct consequence of Theorem 1 that the choice of the path taken by the algorithm to reach location s has no effect on the values of $assignments(s)$ and $pool(s)$ (assuming that the algorithm produces a correct allocation).

A status is assigned to each location which will advance through the following sequence: *Unseen*, *Seen*, and *Visited*. Whenever a location is visited for the first time, all its immediate successors are annotated, and thus become *Seen*. A location is visited only after it has been annotated (and its status is *Seen*). To start the process and establish the invariants, the algorithm begins by annotating the initial location. We assume the existence of the following two functions:

- $reachable : Q \to 2^Q$ maps location q to the set of locations that are reachable from q by some non-empty path.
- $reachable\_from : Q \to 2^Q$ maps location q to the set of locations from which it can be reached by some non-empty path.

The general clock allocation is implemented by Algorithm 4, which consists of a collection of procedures. To make the algorithm more efficient, for every clock t_i of $\mathcal{A}$, the set of *potentially* problematic locations with respect to i, i.e., $pp(i)$, is computed in advance by Algorithm 3.

Algorithm 3: Identifying potentially problematic locations

Input : A timed automaton $\mathcal{A} = \langle E, Q, \{q^0\}, Q_f, V, R\rangle$, function *active*, and the set N of clock numbers.
Output: A mapping from each clock number to the set of potentially problematic locations with respect to that clock number.

foreach $j \in N$ **do**
  $pp(j) := \emptyset$;
foreach $q \in Q$ **do**
  foreach j *such that there are two different transitions* $r, r' \in in(q)$ *where* $j \in active(r) \cap active(r')$ **do**
    $pp(j) := pp(j) \cup \{q\}$;

Algorithm 4: allocating clocks to a general automaton in TA_{DS}

Input : A timed automaton $\mathcal{A} = \langle E, Q, \{q^0\}, Q_f, V, R\rangle$, the initial pool of available clocks $\mathcal{C}$, and functions *active* and *born*.
Output: Annotations for locations.
foreach *location* $s \in Q$ **do**
  Set the status of s to *Unseen*;
$annotate(q^0, \mathcal{C}, \emptyset)$;
$visit(q^0)$;

Procedure annotate(location q, set of clocks p, set of assignments a)

```
/* Invoked only when status of q is Unseen. */
```
$pool(q) := p$;
$assignments(q) := a$;
Set the status of q to *Seen*;

Procedure visit(location q)

```
/* Invoked when the status of q is Seen or Visited. */
```
if *status of* q *is not Visited* **then**
  Set the status of q to *Visited*;
  annotate-immediate-successors(q);
  foreach $r \in out(q)$ **do**
    $visit(target(r))$;

After the clock allocation algorithm has terminated, the value of $alloc.r$, for every transition r, is given by $assignments(target(r))$. The information is easily used to generate new resets and rename clocks in the constraints.

The time complexity is quadratic in the size of the graph.

The resulting allocation is obviously complete. Also, the algorithm satisfies the following theorem:

Procedure annotate-immediate-successors(location q)

Partition $out(q)$ into *mothers* and *others*;
foreach $r \in others$ **do**
  if *status of target*(r) *is Unseen* **then**
    propagate$(q, r, \emptyset)$;
  `/* Otherwise target(r) is already annotated: Theorem 1. */`
if $mothers \neq \emptyset$ **then**
  $Groups :=$ *partition-into-a-set-of-groups*$(q, mothers)$;
  foreach $group \in Groups$ **do**
    $c :=$ *find-clock*$(q, group)$;
    foreach $r \in group$ **do**
      `/* The target of r is Unseen: dominance assumption. */`
      propagate$(q, r, \{c\})$;

Procedure find-clock(location q, set of transitions *group*)

`/* Find a clock for L(q) on transitions in group. */`
$live\_on\_entry := \{j \mid (c, j) \in assignments(q)\}$;
$dying\_all := \bigcap_{r \in group}(live\_on\_entry \setminus active(r))$;
`/* Clocks whose ranges end in all the transitions in group: */`
$released\_all := \{c \mid (c, j) \in assignments(q) \wedge j \in dying\_all\}$;
$available := released\_all \cup pool(q)$;
Return the clock with the smallest number in *available*;

Procedure propagate(location q, transition r, set of clocks sc)

```
/* Invoked only when the target of r is Unseen. q is the source
   of r. Propagate assignments(q) and pool(q) to target(r),
   taking into account that some clock ranges may end on r. If sc
   is not empty, it must be a singleton: assign its member to
   clock number L(q). */
```
$freed\_assignments := \{(d, j) \mid (d, j) \in assignments(q) \wedge j \notin active(r)\}$;
$freed\_clocks := \{d \mid (d, j) \in freed\_assignments\}$;
$tmp\_pool := pool(q) \cup freed\_clocks$;
$tmp\_assignments := assignments(q) \setminus freed\_assignments$;
if $sc \neq \emptyset$ **then**
  $tmp\_pool := tmp\_pool \setminus sc$;
  $tmp\_assignments := tmp\_assignments \cup \{(c, L(q))\}$, where $c \in sc$;
annotate$(target(r), tmp\_pool, tmp\_assignments)$;

Theorem 4 *The computed allocation is correct and lean.*

Proof. All the paths for a clock t_j begin at the same location. The initial transitions of these paths (the "mother" transitions) are partitioned into groups. The members of a group are exactly the initial transitions of a family for t_j. The

Procedure partition-into-a-set-of-groups(location q, set of transitions *mothers*)

```
mother_targets := {target(r) | r ∈ mothers};
/* Initially, each mother is in its own group. */
Groups := ∅;
foreach r ∈ mothers do
    Groups := Groups ∪ {r};
/* Those locations in pp(L(q)) that can be reached from more than
   one mother are the problematic locations. */
foreach s ∈ pp(L(q)) do
    targets := reachable_from(s) ∩ mother_targets;
    Merge those members of Groups that contain transitions whose target is
      in targets;
return Groups;
```

algorithm associates some clock with a group: the association is propagated to all the transitions of the paths for t_j that begin in the group, therefore there is no pair of transitions that satisfies the definition of inconsistency.

When some clock c is assigned to j on the transitions of a group, t_j is the only clock that is born, so c is not, on those transitions, assigned to any i such that $i \neq j$. Moreover, after c is assigned to j, it is removed from the pool and returned only on transitions on which t_j is not active. Therefore c cannot be assigned to any other i on any transition r such that $j \in \mathit{active}(r)$. So the allocation is always an injective function, i.e., it is lean. □

Lemma 2 *Assume alloc is a complete, correct and lean allocation. Then, for any transition r, the number of clocks in alloc.r is not greater than the size of the largest cluster to which r belongs.*

Proof. Assume cluster $\mathcal{F}$ with size n is the largest cluster to which r belongs. Suppose that $|\mathit{alloc}.r| > n$. By Observation 1, $|\mathit{active}(r)| = n' > n$. By Observation 3, the number of families to which r belongs is n'. This implies that r belongs to a cluster $\mathcal{F}'$ whose size is n', which contradicts the assumption. □

Theorem 5 *The computed allocation is optimal.*

Proof. This is a direct consequence of Lemma 2, Theorem 3, Theorem 4 and the fact that the algorithm always allocates the available clock with the smallest number, i.e., a new clock is added to the set of used clocks only when none of those already in the set will do. □

5.4 Generating Clock Constraints and Clock Resets

The final step of our clock allocation method is to generate clock resets and constraints and assign them to the transitions. This step is performed as follows.

Let $r = (s, q, e, \lambda, \phi) \in R$ be a transition, and let $t_j \sim a$ be a constraint on r. Then the constraint will be transformed into $c \sim a$, where $(c, j) \in \mathit{assignments}(q)$.

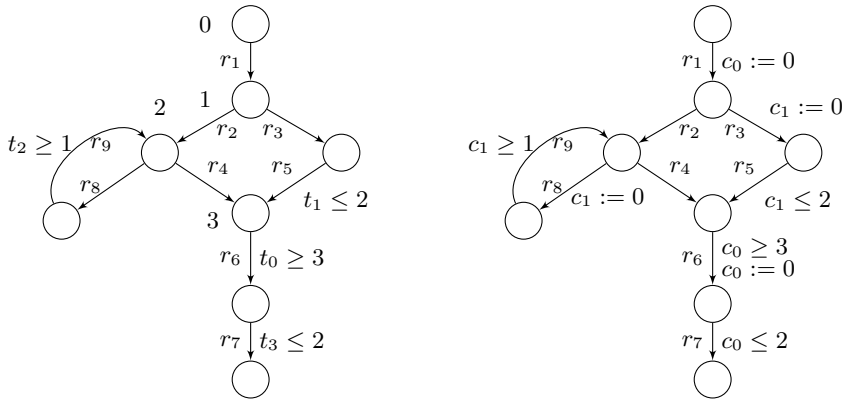

Fig. 4. Two equivalent timed automata

Clock resets are generated by identifying transitions at which ranges begin. For a transition $r = (s, q, e, \lambda, \phi) \in R$, the clock reset $d := 0$ will be added to transition r if $born(r) = \{i\}$ and $(d, i) \in assignments(q)$.

Fig. 4 shows a timed automaton in TA_{DS} along with an equivalent automaton obtained by our clock allocation method: the set of clocks $\{t_0, t_1, t_2, t_3\}$ of the original automaton is replaced by the set $\{c_0, c_1\}$. Observe that the range of clock t_0 that begins at transition r_1 ends on transition r_6 where clock t_3 is born, so the same clock, namely c_0, is assigned to both t_1 and t_3. On transition r_6 the value of clock c_0 is used to check the satisfiability of clock constraint $c_0 \leq 2$ before it is assigned to t_3.

6 Related Work and Conclusions

We study the problem of optimal clock allocation for TA_{DS} (see Definition 1 on p. 4). TA_{DS} is a class of timed automata with some interesting properties. The characteristics of automata belonging to this class allowed us to formulate a particularly efficient clock allocation method, based on liveness analysis of clocks. Given a timed automaton $\mathcal{A} \in TA_{DS}$, the method finds an optimal (in the sense used in our earlier work [**?**]) clock allocation for $\mathcal{A}$ and replaces the original clocks of $\mathcal{A}$ with a new set whose size is minimal, without changing the graph or the form of the constraints in $\mathcal{A}$.

Our approach is different from our earlier method [**?**], in which the optimal clock allocation problem for automata in $TA_S \supsetneq TA_{DS}$ is solved by colouring an interference graph that is obtained from liveness analysis. That method is more general than the one developed in the current paper, as it can be applied to automata in $TA_S \setminus TA_{DS}$. However, our new liveness analysis is much simpler and more efficient, as it takes advantage of the properties of automata in TA_{DS}.

The worst-case complexity of liveness analysis in the cited work [**?**] is $O(|Q|^4)$, where Q is the set of locations. The liveness analysis in the current paper has

the worst-case complexity of $O(|R||N|)$, where R is the set of transitions and $|N|$ is the number of clocks that are *used*.

Being customized for automata in TA_{DS}, the clock allocation algorithm itself is simple and efficient. Even though the interference graph for automata in TA_{DS} is chordal and can be coloured in $O(|R| + |Q|)$, the cost of constructing the interference graph is $O(|Q|^3)$. The complexity of our clock allocation is $O(|Q|^2)$.