Synthesizing Certificates in Networks of Timed Automata*

Bernd Finkbeiner Hans-Jörg Peter Sven Schewe
Saarland University *University of Liverpool*
{*finkbeiner* | *peter*}*@cs.uni-sb.de* *sven.schewe@liverpool.ac.uk*

Abstract

We present an automatic method for the synthesis of certificates for components in embedded real-time systems. A certificate is a small homomorphic abstraction that can transparently replace the component during model checking: if the verification with the certificate succeeds, then the component is guaranteed to be correct; if the verification with the certificate fails, then the component itself must be erroneous. We give a direct construction, based on a forward and backward reachability analysis of the timed system, and an iterative refinement process, which produces a series of successively smaller certificates. In our experiments, model checking the certificate is several orders of magnitude faster than model checking the original system.

1 Introduction

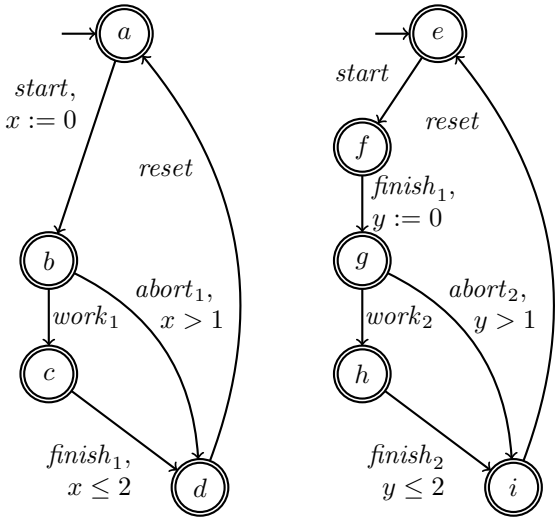
Model checking allows the developer of an embedded real-time system to detect inconsistent timing requirements and functional errors early in the design process. If the system contains an error, tools like UPPAAL [18] provide evidence in the form of an error trace, which can be used to reproduce the problem during simulation. If the system is correct, however, most model checkers only report the fact, without providing evidence that would help the designer understand *why* the system is correct, or help an independent verifier reproduce the proof.

In this paper, we present an automatic abstraction technique which provides such evidence. For a given component in a network of timed automata, we compute a quotient automaton, which we call the component's *certificate*. The certificate satisfies three key properties. First, it is sound to replace the component with its certificate during the verification of the network. We guarantee both that if the verification with the certificate succeeds, then the component itself is correct, and that if the verification with the certificate fails, then the component itself is erroneous. Second, it is easy to verify the validity of the certificate: The certificate is a simple homomorphic abstraction of the component, which means that each location of the certificate represents a set of locations in the component. Hence, verifying that the component is an accurate implementation of the certificate amounts to a simple (syntactic) simulation check. Third, the certificate is much smaller than the component. Since certificates only need to preserve those component properties that are actually necessary to establish the correctness of the full network, they can be based on a coarse equivalence relation. The resulting quotients are small, as illustrated by the following example.

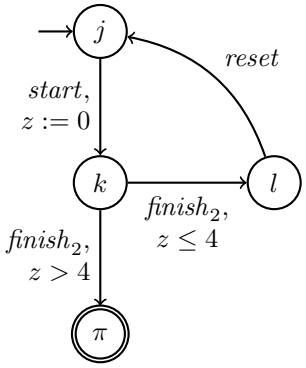
Figure 1 shows a network of timed automata modeling a simple production plant with two controllers. The plant processes workpieces at a rate of up to 0.5 pieces per second. When a workpiece enters the plant, both controllers are notified with the *start* signal. Then, the machine controlled by the first controller works on the piece ($work_1$) and finishes within two seconds. Once the first machine is done ($finish_1$), the machine controlled by the second controller works on the piece ($work_2$) and finishes ($finish_2$) again within at most two seconds. Afterwards, the controllers may be reset (*reset*) to be ready for the next workpiece. The actions $abort_1$ and $abort_2$ model a situation where the respective machine has not started working after one second, in which case the controllers can abort. We are interested in the property that the work on each workpiece is done (if completed) in at most four seconds.

Figure 2 shows the certificate for controller #2. Locations e, f, i and locations g, h have been merged into single locations. Clearly, this abstraction extends the observable behavior of controller #2: for example, the controller now accepts an arbitrary number of *start* signals. However, the certificate is sound for proving that

*This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).



(a) Controller #1 (b) Controller #2



(c) Property automaton

Figure 1. Network of timed automata modeling a simple production plant with two controllers. The following self loops are implicit: $work_2, finish_2$ and $abort_2$ on all locations of Controller #1; $work_1, finish_1$ and $abort_1$ on all locations of Controller #2; $work_1, work_2, abort_1$ and $abort_2$ on all locations of the property automaton.

the work on each piece is done within four seconds, because the relevant requirement for controller #2, that its work is finished within two seconds after receiving the $finish_1$ signal, is preserved.

Our construction of the certificate is based on two equivalence relations over the locations of a given automaton within a network of timed automata. As explained in Section 4, two locations m_1 and m_2 are *forward-equivalent* if the sets of states that can be reached at m_1 and at m_2 are the same; dually, m_1 and m_2 are *backward-equivalent* if the sets of states that, starting at m_1 and at m_2, can reach the error are the same. In the example, locations g and h of controller #2 are forward-equivalent because they are

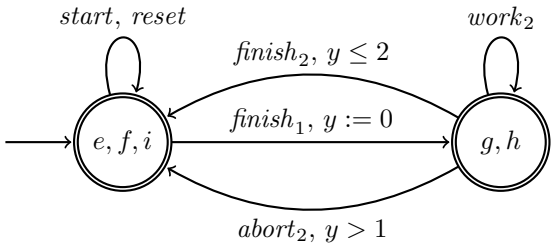


Figure 2. The certificate for controller #2 in the production plant example shown in Figure 1. There are implicit self loops for $work_1, finish_1$ and $abort_1$ on all locations.

both reachable in conjunction with locations d and k and clock values $x, y, z \in \mathbb{R}_{\geq 0}$. Locations e, f, and i are backward-equivalent: the error location π is unreachable from all these locations.

The forward and backward equivalences can be computed directly, by computing the sets of forward and backward reachable states in the network. In Section 6 we additionally show that it is possible to construct the equivalence in an iterative fashion, where, starting with a complete partition of the location set, successively more and more locations are merged. This approach has the advantage that it is possible to interrupt the process as soon as the certificate has become sufficiently small.

In Section 7, we present experimental results that indicate that model checking the certificate is significantly faster (in our experiments, by several orders of magnitude) than model checking the original system.

Related Work. The term *certifying model checkers* was coined by Namjoshi [20] in the setting of μ-calculus model checking for labeled finite-state transition systems. Different from our component-based setting, a certificate in [20] is a deductive proof of a global property, which is checked by inductive, rather than fixpoint-based methods.

Certificate synthesis reduces the size of a timed automaton by merging locations. This approach can be compared to reduction techniques that merge states. Typically, some initial partition of the state space is split until the coarsest stable refinement is reached [2, 21, 11], or states are collapsed based on some equivalence such as history equivalence or transition bisimulation [15]. An early proposal for an equivalence that is parameterized with information about the context of a process is *context dependent process equivalence* [19].

State minimization techniques are useful to obtain a

compact finite representation of the infinite state space of a timed automaton. As systems with dense time have an uncountable state space, all model checking algorithms build on abstraction (and hence on state minimization). The most widespread approach is to use approximate [5] or precise [18, 9] abstractions of the finite region graph [3] of timed automata. A common problem with these abstraction methods is, however, that they are not compositional and therefore cannot be applied to individual automata in a network of timed automata. Other reduction techniques, which can potentially be combined with state minimization, include partial order reduction (based on a local-time semantics) [7] and clock elimination [10].

Algorithms similar to certificate synthesis are studied in the setting of *compositional model checking*. To prove a property $\mathcal{P}$ for the parallel composition $\mathcal{M}\|\mathcal{N}$ of two timed automata $\mathcal{M}$ and $\mathcal{N}$, the compositional model checker CMC [16, 17] first transforms the property with respect to $\mathcal{N}$ into $\mathcal{P}/\mathcal{N}$, and then, after simplification, further into $\mathcal{P}/\mathcal{N}/\mathcal{M}$. The transformed property $\mathcal{P}/\mathcal{N}/\mathcal{M}$ is checked against the unit automaton **1**. In this process, $\mathcal{P}/\mathcal{N}$ can be understood as a certificate for $\mathcal{M}$, because, if $\mathcal{M}$ satisfies $\mathcal{P}/\mathcal{N}$, then $\mathcal{M}\|\mathcal{N}$ must satisfy $\mathcal{P}$. A certificate generated in this way is not guaranteed to be a homomorphic abstraction of $\mathcal{M}$, however. In fact, the computation of $\mathcal{P}/\mathcal{N}$ is completely independent of $\mathcal{M}$.

A prominent approach to the compositional model checking of *untimed systems* is by *learning* certificates as deterministic word automata [8, 4, 1]. Here, a preliminary certificate $\mathcal{C}$ (initially, an automaton accepting the full language) is evaluated against both $\mathcal{N}$ and $\mathcal{P}$ by model checking. As long as either $\mathcal{C}$ rejects some computation of $\mathcal{N}$ or $\mathcal{M}\|\mathcal{C}$ accepts a computation that violates $\mathcal{P}$, $\mathcal{C}$ is refined to eliminate the particular counterexample. This approach has been successful for discrete systems (cf. the *LTSA* tool [8]). Since no similar learning algorithms are known for timed languages, however, an immediate extension to real-time systems appears impossible.

As a preparatory step to the work presented in this paper, we investigated quotient-based certificates in the discrete setting of the SPIN model checker [14]. Given two Promela processes $\mathcal{M}, \mathcal{N}$ and a property automaton $\mathcal{P}$, our tool RESY [13, 12] performs a graph-theoretic analysis of the product of $\mathcal{N}$ and $\mathcal{P}$ to identify states in $\mathcal{M}$ that can safely be merged. For timed systems, a graph-theoretical analysis alone is, of course, not sound, because one location may be safe and another unsafe, even if both have a (discrete) path to an error location.

Contribution. In this paper, we present a general theory and algorithms for the synthesis of certificates in networks of timed automata. The contributions of the paper are the following.

- We define novel equivalence relations for timed automata, which are coarser than simulation but still sound for compositional model checking.
- Based on the new equivalence relations, we present an algorithm for the automatic synthesis of certificates.
- We present an incremental approach for the synthesis of certificates, which can be interrupted at any time to produce a sound intermediate certificate.

2 Preliminaries

Timed Automata. A *timed automaton* [3] is a tuple $\mathcal{A} = (L, I, \Sigma, \Delta, \chi, F)$, where L is a finite set of locations, $I \subseteq L$ is a set of the initial locations, Σ is a finite set of actions, $\Delta \subseteq (L \times \Sigma \times \mathcal{C}(\chi) \times 2^{\chi} \times L)$ is a transition relation, χ is a finite set of real valued clocks, and $F \subseteq L$ is a set of final locations.

The clock constraints $\varphi \in \mathcal{C}(\chi)$ are of the form

$$\varphi = x \leq c \mid c \leq x \mid x < c \mid c < x \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2,$$

where x is a clock in χ and c is a constant in $\mathbb{N}_0$. A *clock valuation* $\vec{t} : \chi \to \mathbb{R}_{\geq 0}$ assigns a non-negative value to each clock and can also be represented by a $|\chi|$-dimensional vector $\vec{t} \in \mathbb{R}^{\chi}_{\geq 0}$. We use $\mathcal{R} = 2^{\mathbb{R}^{\chi}_{\geq 0}}$ to denote the set of all clock valuations.

The states of a timed automaton are pairs $(l, \vec{t})$ of locations and clock valuations. Timed automata have two types of transitions: *timed transitions*, where only time passes and the location remains unchanged, and *discrete transitions* Δ. A timed transition, denoted by $(l, \vec{t}) \xrightarrow{a} (l, \vec{t} + a \cdot \vec{1})$, consists of adding the same non-negative value $a \in \mathbb{R}_{\geq 0}$ to all clocks. A discrete transition, denoted $(l, \vec{t}) \xrightarrow{a} (l', \vec{t}')$ for some $a \in \Sigma$, is a transition $\delta = \langle l, a, \varphi, \lambda, l' \rangle$ of Δ such that $\vec{t}$ satisfies the clock constraint φ of δ, and $\vec{t}' = \vec{t}[\lambda := 0]$ is obtained from $\vec{t}$ by setting the clocks in λ to 0.

We distinguish *system automata,* which only have final locations, from *property automata*, where the set of final locations forms a proper subset of the locations. We assume that, in a property automaton, the sets of initial and final locations are disjoint.

We say that a finite sequence $a_1 \dots a_n \in (\Sigma \cup \mathbb{R}_{\geq 0})^*$ of transitions is in the *language of* $\mathcal{A}$ ($a_1 \dots a_n \in \mathcal{L}(\mathcal{A})$) if there is a path $s_0 \xrightarrow{a_1} s_1 \dots s_{n-1} \xrightarrow{a_n} s_n$ such that

the single $s_i = (l_i, \vec{t}_i)$ are states of the automaton, s_0 is an initial state (that is, $l_0 \in I$ is an initial location and $\vec{t}_0 = \vec{0}$ is the zero vector), and $s_{i-1} \xrightarrow{a_i} s_i$ are transitions of $\mathcal{A}$. We write $s_0 \longrightarrow^* s_n$ for the existence of a finite sequence $a_1 \dots a_n \in (\Sigma \cup \mathbb{R}_{\geq 0})^*$ of transitions with $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n$, and call a finite automaton *safe* if no final state is reachable from an initial state ($\nexists i \in I,\ f \in F,\ \vec{t} \in \mathcal{R}.\, (i, \vec{0}) \longrightarrow^* (f, \vec{t})$).

Composition. Timed automata can be composed to networks, in which the automata run in parallel and synchronize on shared actions. For two timed automata $\mathcal{A}_1 = (L_1, I_1, \Sigma_1, \Delta_1, \chi_1)$ and $\mathcal{A}_2 = (L_2, I_2, \Sigma_2, \Delta_2, \chi_2)$ with disjoint clock sets $\chi_1 \cap \chi_2 = \emptyset$, the *parallel composition* $\mathcal{A}_1 \| \mathcal{A}_2$ is the timed automaton $(L_1 \times L_2, I_1 \times I_2, \Sigma_1 \cup \Sigma_2, \Delta, \chi_1 \cup \chi_2)$, where Δ is the smallest set that contains

- for $a \in \Sigma_1 \cap \Sigma_2$, $\langle (l_1, l_2), a, \varphi_1 \wedge \varphi_2, \lambda_1 \cup \lambda_2, (l_1', l_2') \rangle$ if $\langle l_1, a, \varphi_1, \lambda_1, l_1' \rangle \in \Delta_1$ and $\langle l_2, a, \varphi_2, \lambda_2, l_2' \rangle \in \Delta_2$,
- for $a \in \Sigma_1 \setminus \Sigma_2$, $\langle (l_1, l_2), a, \varphi_1, \lambda_1, (l_1', l_2) \rangle$ if $\langle l_1, a, \varphi_1, \lambda_1, l_1' \rangle \in \Delta_1$, and
- for $a \in \Sigma_2 \setminus \Sigma_1$, $\langle (l_1, l_2), a, \varphi_2, \lambda_2, (l_1, l_2') \rangle$ if $\langle l_2, a, \varphi_2, \lambda_2, l_2' \rangle \in \Delta_2$.

For the ease of argumentation, we assume that all timed automata within a network have the same set Σ of actions. Technically, we can complete a timed automaton by adding a transition $\langle l, a, \mathit{true}, \emptyset, l \rangle$ for every additional symbol a and every location l of the timed automaton without changing the network semantics. This completion implies that $(s_1^1, s_1^2; \vec{t}_1^1, \vec{t}_1^2) \xrightarrow{a_1} (s_2^1, s_2^2; \vec{t}_2^1, \vec{t}_2^2) \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} (s_n^1, s_n^2; \vec{t}_n^1, \vec{t}_n^2)$ is a path in $\mathcal{A}_1 \| \mathcal{A}_2$ if, and only if, $(s_1^1, \vec{t}_1^1) \xrightarrow{a_1} (s_2^1, \vec{t}_2^1) \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} (s_n^1, \vec{t}_n^1)$ is a path in $\mathcal{A}_1$ and $(s_1^2, \vec{t}_1^2) \xrightarrow{a_1} (s_2^2, \vec{t}_2^2) \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} (s_n^2, \vec{t}_n^2)$ is a path in $\mathcal{A}_2$. In particular, $\mathcal{A}_1 \| \mathcal{A}_2$ is safe if there is no path on which a final state is reachable in $\mathcal{A}_1$ and $\mathcal{A}_2$ at the same time.

Finite Representation. The decidability of timed automata relies on the possibility to symbolically represent the unbounded semantics in the finite *region graph* [3], which in turn can be represented efficiently by federations of clock zones [6].

For a timed automaton $\mathcal{A} = (L, I, \Sigma, \Delta, \chi, F)$, we call the value of a clock $x \in \chi$ *maximal* if it is strictly greater than the highest constant c_{max} any clock is compared to. (c_{max} is sometimes called the clock ceiling.) We say that two clock valuations $\vec{t}_1, \vec{t}_2 : \chi \to \mathbb{R}_{\geq 0}$ are in the same *clock region*, denoted $\vec{t}_1 \sim_R \vec{t}_2$, if

- the set of clocks with maximal value is the same in $\vec{t}_1$ and in $\vec{t}_2$ ($\forall x \in \chi.\, \vec{t}_1(x) > c_{max} \Leftrightarrow \vec{t}_2(x) > c_{max}$), and
- $\vec{t}_1$ and $\vec{t}_2$ agree (1) on the integer parts of the clock values, (2) on the relative order of the non-integer parts of the clock values, and (3) on the equality of the non-integer parts of the clock values with 0. That is, for all clocks x, y with non-maximal value, it holds that (1) $\lfloor \vec{t}_1(x) \rfloor = \lfloor \vec{t}_2(x) \rfloor$, (2) $\widehat{\vec{t}}_1(x) \leq \widehat{\vec{t}}_1(y) \Leftrightarrow \widehat{\vec{t}}_2(x) \leq \widehat{\vec{t}}_2(y)$, and (3) $\widehat{\vec{t}}_1(x) = 0$ if, and only if, $\widehat{\vec{t}}_2(x) = 0$, where $\widehat{\vec{t}}_i(x) = \vec{t}_i(x) - \lfloor \vec{t}_i(x) \rfloor$ for $i = 1, 2$.

We denote with $[\vec{t}]_R = \{\vec{t}' \in \mathcal{R} \mid \vec{t} \sim_R \vec{t}'\}$ the clock region $\vec{t}$ belongs to. We say that two states $s_1 = (l_1, \vec{t}_1)$ and $s_2 = (l_2, \vec{t}_2)$ of $\mathcal{A}$ are *region-equivalent*, denoted by $s_1 \sim_R s_2$, if their locations are the same ($l_1 = l_2$) and the clock valuations are in the same clock region ($\vec{t}_1 \sim_R \vec{t}_2$), and denote with $[(l, \vec{t})]_R = \{(l, \vec{t}') \in L \times \mathcal{R} \mid \vec{t} \sim_R \vec{t}'\}$ the equivalence class of region-equivalent states $(l, \vec{t})$ belongs to.

Regions are a suitable semantics for the abstraction of timed automata, because they essentially preserve the language: if there is a discrete transition $s \xrightarrow{a} s'$ from a state s to a state s' of a timed automaton, then there is, for all states $r \sim_R s$ region-equivalent to s, a state $r' \sim_R s'$ region-equivalent to s', such that $r \xrightarrow{a} r'$ is a discrete transition with the same label. For timed transitions, a slightly weaker property holds: If there is a timed transition $s \xrightarrow{t} s'$ from a state s to a state s', then there is, for all states $r \sim_R s$ region-equivalent to s, a state $r' \sim_R s'$ region-equivalent to s' such that there is a timed transition $r \xrightarrow{t'} r'$ (but possibly with a different $t' \neq t$).

The *finite semantics* of a timed automaton $\mathcal{A} = (L, I, \Sigma, \Delta, \chi, F)$ is the finite graph $\mathsf{sem}(\mathcal{A}) = (S, I', \Sigma, \Delta', \chi, F')$ where

- the abstract states $S = \{[(l, \vec{t})]_R \mid (l, \vec{t}) \in L \times \mathcal{R}\}$ of $\mathsf{sem}(\mathcal{A})$ is the set of equivalence classes of region-equivalent states of $\mathcal{A}$, with
- the classes $I' = (I \times \{\vec{0}\})/_{\sim_R}$ of states that are region-equivalent to initial states of $\mathcal{A}$ as initial states,
- the set $\Delta' = \{(s, s') \in S \times S \mid \exists r \in s,\ r' \in s', a \in \Sigma \cup \mathbb{R}_{\geq 0}.\, r \xrightarrow{a} r'\}$ of transitions, and
- the classes $F' = \{[(l, \vec{t})]_R \mid (l, \vec{t}) \in F \times \mathcal{R}\}$ of states that are region-equivalent to final states of $\mathcal{A}$ as final states.

We denote $(s, s') \in \Delta'$ by $s \to s'$. The finite semantics is safety preserving:

Lemma 2.1 *[3] For a timed automaton $\mathcal{A} = (L, I, \Sigma, \Delta, \chi, F)$ there is a finite path from a state $(l, \vec{t})$ to a state $(l', \vec{t}')$ if, and only if, there is a finite path from $\left[(l, \vec{t})\right]_R$ to $\left[(l', \vec{t}')\right]_R$ in $\mathsf{sem}(\mathcal{A})$.*

Proof: For every finite path $s_0 \xrightarrow{a_1} s_1 \dots s_{n-1} \xrightarrow{a_k} s_k$ of $\mathcal{A}$, $[s_0]_R \to [s_1]_R \dots [s_{n-1}]_R \to [s_k]_R$ is a path in $\mathsf{sem}(\mathcal{A})$ by the definition of $\mathsf{sem}(\mathcal{A})$.

Conversely, we show by induction on the length of the path that, for every path in $\mathsf{sem}(\mathcal{A})$ from $\left[(l, \vec{t})\right]_R$ to $\left[(l', \vec{t}')\right]_R$, $\mathcal{A}$ has a path from $(l, \vec{t})$ to a state $s \sim_R (l', \vec{t}')$ region equivalent to $(l', \vec{t}')$. A transition $[s_1] \to [s_2]$ in the finite semantics implies that there are representatives $r_1 \sim_R s_1$ and $r_2 \sim_R s_2$ of $[s_1]_R$ and $[s_2]_R$, respectively, and an $a \in \Sigma \cup \mathbb{R}_{\geq 0}$, such that $r_1 \xrightarrow{a} r_2$ holds true. We distinguish two cases:

(1) If this concrete transition is discrete ($a \in \Sigma$) than it refers to some transition $\delta = \langle l, a, \varphi, \lambda, l' \rangle$. δ can be taken from all representatives $(l, \vec{t}_1) \sim_R s_1$ of $[s_1]_R$ because the validity of φ is independent of the representative. Taking δ from two representatives $(l, \vec{t}_1), (l, \vec{t}_1') \sim_R s_1$ of $[s_1]_R$ leads to states $(l', \vec{t}_2)$ and $(l', \vec{t}_2')$ with the same location and $\vec{t}_2 = \vec{t}_1[\lambda := 0]$ and $\vec{t}_2' = \vec{t}_1'[\lambda := 0]$. Since $\vec{t}_1 \sim_R \vec{t}_1'$ implies $\vec{t}_1[\lambda := 0] \sim_R \vec{t}_1'[\lambda := 0]$, $(l', \vec{t}_2) \sim_R (l', \vec{t}_2')$ holds true.

(2) If this concrete transition is timed ($a \in \mathbb{R}_{\geq 0}$), it suffices to show that $\vec{t}_1 \sim_R \vec{t}_1'$ and $\vec{t}_2 = \vec{t}_1 + a \cdot \vec{1}$ implies the existence of an $a' \in \mathbb{R}_{\geq 0}$ such that $\vec{t}_2' = \vec{t}_1' + a' \cdot \vec{1}$. This is obviously true: If one of the non-maximal clocks, say x, has an integer value in $\vec{t}_2$, we have to choose $a' = \vec{t}_2(x) - \vec{t}_1'(x)$. Otherwise we pick a non-maximal clock x with a minimal fractional part f and set $a' = \vec{t}_2(x) - f - \vec{t}_1'(x) + \varepsilon$ for a sufficiently small ε. (Sufficiently small means smaller than all strictly positive fractional parts of clock values and of differences of clock values.) □

3 The Certificate Synthesis Problem

We now give a formal definition for the problem of synthesizing certificates in networks of timed automata. Let $\mathcal{M}$ be a timed automaton in a network $\mathcal{M} \| \mathcal{N}$. We call the timed automaton $\mathcal{N}$ the *environment* of $\mathcal{M}$. Typically, $\mathcal{N}$ is the parallel composition of several system automata and some property automaton that defines the safety-critical properties of the complete network.

A timed automaton $\mathcal{C}$ is a *certificate* for $\mathcal{M}$ in $\mathcal{M} \| \mathcal{N}$ if $\mathcal{C}$ is a sound homomorphic abstraction of $\mathcal{M}$. Sound homomorphic abstractions are defined as follows:

- A timed automaton $\mathcal{M}'$ is a *homomorphic abstraction* of a timed automaton $\mathcal{M} = (L, I, \Sigma, \Delta, \chi, F)$, if there exists an equivalence relation $\simeq \subseteq L \times L$ on the locations of $\mathcal{M}$ such that $\mathcal{M}'$ is the quotient of $\mathcal{M}$ with respect to $\simeq$. For a given equivalence relation $\simeq$, the quotient $\mathcal{M}/_{\simeq}$ is defined as the timed automaton $(L', I', \Sigma, \Delta', \chi, F')$ with
  - $L' = \{[l] \mid l \in L\}$ where $[l] = \{l' \mid l' \simeq l\}$ denotes the equivalence class of a location $l \in L$ with respect to $\simeq$,
  - $I' = \{[l] \mid l \in I\}$, $F' = \{[l] \mid l \in F\}$, and
  - $\Delta' = \{\langle [l], a, \varphi, \lambda, [l'] \rangle \mid \langle l, a, \varphi, \lambda, l' \rangle \in \Delta\}$.
- A timed automaton $\mathcal{M}'$ is a *sound abstraction* of a timed automaton $\mathcal{M}$ in a network $\mathcal{M} \| \mathcal{N}$, if it holds that $\mathcal{M} \| \mathcal{N}$ is safe if and only if $\mathcal{M}' \| \mathcal{N}$ is safe.

In general, a timed automaton $\mathcal{M}$ may have multiple certificates; in particular, $\mathcal{M}$ itself is always a certificate, where the equivalence $\simeq$ is simply the identity relation on the locations. Computing the *minimal* certificate is possible in theory (for example, by enumerating all certificates) but too expensive in practice:

Theorem 3.1 *For a timed automaton $\mathcal{M}$ in a network $\mathcal{M} \| \mathcal{N}$ and a positive integer k, the problem of deciding whether there exists a certificate for $\mathcal{M}$ with k locations is NP-complete in the number of locations of $\mathcal{M} \| \mathcal{N}$.*

Proof: Safety of $\mathcal{M} \| \mathcal{N}$ can be checked in linear time in the number of locations; the problem is therefore in NP. We show NP-hardness with a reduction from graph k-colorability. An undirected graph $G = (V, E)$ is k-colorable if there is a function $f : V \to \{1, 2, \dots, k\}$ such that $f(u) \neq f(v)$ whenever there is an edge $\{u, v\} \in E$. Let $V = \{v_1, \dots, v_n\}$. To decide k-colorability of G, we consider the following pair of timed automata $\mathcal{M}, \mathcal{N}$. The automaton $\mathcal{M} = (V, V, E, \Delta_{\mathcal{M}}, \emptyset, V)$ has one location for each vertex in V. The actions consist of the edges in E. For each action $\{v_i, v_j\}$ we add a transition from location v_i to location v_j if $i < j$: $\Delta_{\mathcal{M}} = \{(v_i, \{v_i, v_j\}, \mathit{true}, \emptyset, v_j) \mid v_i, v_j \in V, i < j\}$. The automaton $\mathcal{N} = (E \cup \{s_I, s_F\}, \{s_I\}, E, \Delta_E, \emptyset, \{s_F\})$ reaches the final location s_F only on paths with exactly two discrete transitions that have the same action. We add a transition from the initial location on input e to location e, and from location e on input e to the final location s_F: $\Delta_E = \{(s_0, e, \mathit{true}, \emptyset, e) \mid e \in E\} \cup \{e, e, \mathit{true}, \emptyset, s_F) \mid e \in E\}$.

On the one hand, every certificate $\mathcal{C}$ for $\mathcal{M}$ in $\mathcal{M} \| \mathcal{N}$, whose equivalence $\simeq$ has k equivalence classes, defines

a k-coloring f of G: $(v_i \simeq v_j) \Leftrightarrow (f(v_i) = f(v_j))$: if there were a pair of vertices v_i, v_j with $\{v_i, v_j\} \in E$ and $v_i \simeq v_j$, then $\mathcal{C}\|\mathcal{N}$ would have an error path on $\{v_i, v_j\}, \{v_i, v_j\}$, whereas $\mathcal{M}\|\mathcal{N}$ does not have any error paths. On the other hand, if G is k-colorable, then the quotient of $\mathcal{M}$ with respect to $\simeq$ is a certificate, because $\mathcal{C}\|\mathcal{N}$, like $\mathcal{M}\|\mathcal{N}$, has no error paths: on every path in $\mathcal{C}\|\mathcal{N}$, each $e \in E$ occurs at most once. □

In the following sections we present equivalence relations that, while inexpensive to compute, define small certificates.

4 Forward and Backward Equivalences

In this section, we define the *forward equivalence* $\simeq_F$ and the *backward equivalence* $\simeq_B$ over the locations of a timed automaton $\mathcal{M}$ in a network $\mathcal{M}\|\mathcal{N}$. Intuitively, two locations of $\mathcal{M}$ are forward-equivalent, if merging them does not make additional states reachable in $\mathsf{sem}(\mathcal{M}\|\mathcal{N})$, and backward-equivalent, if merging them does not make final states reachable from additional states in $\mathsf{sem}(\mathcal{M}\|\mathcal{N})$.

Let $L_\mathcal{M}$ and $L_\mathcal{N}$ be the locations of $\mathcal{M}$ and $\mathcal{N}$, respectively, and let $\mathsf{sem}(\mathcal{M}\|\mathcal{N}) = (L_\mathcal{M} \times L_\mathcal{N} \times \mathcal{R}/_{\sim_R}, I, \Sigma, \Delta, \chi, F)$. For locations $m_1, m_2 \in L_\mathcal{M}$ we define

$$\begin{aligned} m_1 \simeq_F m_2 \Leftrightarrow \forall n \in L_\mathcal{N}, \vec{t} \in \mathcal{R}.& \\ \exists i_1 \in I \text{ s.t. } & i_1 \longrightarrow^* \left[(m_1, n, \vec{t})\right]_R \\ \Leftrightarrow \exists i_2 \in I \text{ s.t. } & i_2 \longrightarrow^* \left[(m_2, n, \vec{t})\right]_R; \\ m_1 \simeq_B m_2 \Leftrightarrow \forall n \in L_\mathcal{N}, \vec{t} \in \mathcal{R}.& \\ \exists f_1 \in F \text{ s.t. } & \left[(m_1, n, \vec{t})\right]_R \longrightarrow^* f_1 \\ \Leftrightarrow \exists f_2 \in F \text{ s.t. } & \left[(m_2, n, \vec{t})\right]_R \longrightarrow^* f_2. \end{aligned}$$

Both equivalences define certificates.

Theorem 4.1 *For a timed automaton $\mathcal{M}$ in a network $\mathcal{M}\|\mathcal{N}$, both $\mathcal{M}/_{\simeq_F}$ and $\mathcal{M}/_{\simeq_B}$ are certificates of $\mathcal{M}$.*

Proof: We prove that a final state is reachable in $\mathcal{M}\|\mathcal{N}$ if, and only if, a final state is reachable in $\mathcal{M}/_{\simeq_F}\|\mathcal{N}$ and $\mathcal{M}/_{\simeq_B}\|\mathcal{N}$, respectively. For the *"if"* direction, consider an arbitrary path of $\mathcal{M}\|\mathcal{N}$ to a final state: $(m_0, n_0, \vec{0}) \xrightarrow{a_1} (m_1, n_1, \vec{t}_1) \xrightarrow{a_2} \ldots \xrightarrow{a_n} (m_k, n_k, \vec{t}_n)$. Then there exists the corresponding path $([m_0], n_0, \vec{0}) \xrightarrow{a_1} ([m_1], n_1, \vec{t}_1) \xrightarrow{a_2} \ldots \xrightarrow{a_n} ([m_k], n_k, \vec{t}_n)$ in $\mathcal{M}/_{\simeq_F}\|\mathcal{N}$ and $\mathcal{M}/_{\simeq_B}\|\mathcal{N}$, respectively.

To prove the *"only if"* direction for forward equivalence, we first observe that the reachability of $(m, n, [\vec{t}]_R)$ in $\mathsf{sem}(\mathcal{M}\|\mathcal{N})$ implies by the definition of forward equivalence that, for all states $m' \in [m]_F$ forward equivalent to $m \simeq_F m'$, $(m', n, [\vec{t}]_R)$ is reachable in $\mathsf{sem}(\mathcal{M}\|\mathcal{N})$, too.

Using this observation, we show that the existence of a path from an initial state to a final state $([m]_F, n, [\vec{t}]_R)$ in $\mathsf{sem}(\mathcal{M}/_{\simeq_F}\|\mathcal{N})$ implies the existence of a path from an initial state to a final state $(m', n, [\vec{t}]_R)$ for all representatives $m' \simeq_F m$ of $[m]_F$ in $\mathsf{sem}(\mathcal{M}\|\mathcal{N})$ by induction over the length of the path in $\mathsf{sem}(\mathcal{M}/_{\simeq_F}\|\mathcal{N})$.

For the induction basis, the claim holds true for traces of length 0: If $([m]_F, n, [\vec{t}]_R)$ is forward-reachable by a trace of length 0 in $\mathsf{sem}(\mathcal{M}/_{\simeq_F}\|\mathcal{N})$, then $([m]_F, n, [\vec{t}]_R) \in I$ is initial. (That is, n and a representative $i \sim m$ of $[m]_F$ are initial locations of $\mathcal{N}$ and $\mathcal{M}$, respectively, and $\vec{t} = \vec{0}$.) Using the previous observation, this implies that $s' = (m', n, [\vec{t}]_R)$ is forward-reachable for every location $m' \simeq_F m$ that is forward-equivalent to m.

For the induction step $(k \mapsto k+1)$, assume that $([m_0]_F, n_0, [\vec{0}]_R) \to ([m_1]_F, n_1, [\vec{t}_1]_R) \to \ldots \to ([m_k]_F, n_k, [\vec{t}_n]_R) \to ([m_{k+1}]_F, n_{k+1}, [\vec{t}_{k+1}]_R)$ and that $(e_0, [m_0]_F, [\vec{0}]_R)$ is an initial state of $\mathsf{sem}(\mathcal{M}/_{\simeq_F}\|\mathcal{N})$. By induction hypothesis, all representatives $s_k = (m', n_k, [\vec{t}_n]_R)$ in $\mathsf{sem}(\mathcal{M}\|\mathcal{N})$ of $([m_k]_F, n_k, [\vec{t}_n]_R)$ in $\mathsf{sem}(\mathcal{M}/_{\simeq_F}\|\mathcal{N})$, there is finite trace from some initial state i of $\mathsf{sem}(\mathcal{M}\|\mathcal{N})$ to s_k ($i \longrightarrow^* s_k$). By the definition of homomorphic abstractions, $([m_k]_F, n_k, [\vec{t}_n]_R) \to ([m_{k+1}]_F, n_{k+1}, [\vec{t}_{k+1}]_R)$ implies that there are representatives $m' \in [m_k]$ and $m'' \in [m_{k+1}]_F$ such that $(m', n_k, [\vec{t}_n]_R) \to (m'', n_{k+1}, [\vec{t}_{k+1}]_R)$ is a transition of $\mathsf{sem}(\mathcal{M}\|\mathcal{N})$. Together, this implies that $(m'', n_{k+1}, [\vec{t}_{k+1}]_R)$ is reachable in $\mathsf{sem}(\mathcal{M}\|\mathcal{N})$, and, using the previous observation, we can conclude that *all* representatives of $([m'']_F, n_{k+1}, [\vec{t}_{k+1}]_R)$ are reachable in $\mathsf{sem}(\mathcal{M}\|\mathcal{N})$.

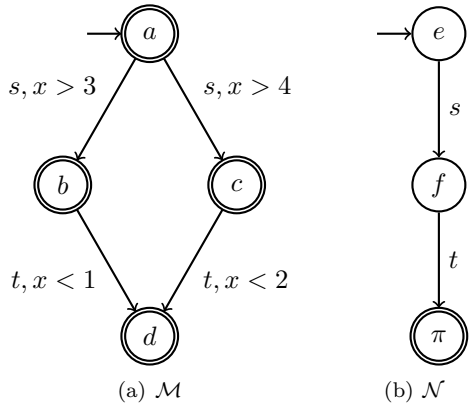
The *"only if"* direction for backward equivalence can be demonstrated analogously. □

The computation of the set of reachable states is a standard fixed point construction. Let $(S, I, \Sigma, \Delta, X, F) = \mathsf{sem}(\mathcal{M}\|\mathcal{N})$ be the finite semantics of the composition of $\mathcal{M}$ and $\mathcal{N}$.

- $\mathsf{Succ}(S') = \{s \in S \mid \exists s' \in S'.\, s' \to s\}$, and
- $\mathsf{Pred}(S') = \{s \in S \mid \exists s' \in S'.\, s \to s'\}$,

that map a set S' of states to the states reachable from some state in S' and from which some state in S' is reachable, respectively, then the set FR of forward-reachable states and the set BR of backward reachable states are obtained by the following fixed point computations:

$$\begin{array}{ll|ll} \mathsf{FR}_0 & = I & \mathsf{BR}_0 & = F \\ \mathsf{FR}_{i+1} & = \mathsf{Succ}(\mathsf{FR}_i) & \mathsf{BR}_{i+1} & = \mathsf{Pred}(\mathsf{BR}_i) \\ \mathsf{FR} & = \lim_i \mathsf{FR}_i & \mathsf{BR} & = \lim_i \mathsf{BR}_i. \end{array}$$



(a) $\mathcal{M}$ (b) $\mathcal{N}$

Figure 3. Example network $\mathcal{M}\|\mathcal{N}$. Locations b and c are neither *forward* nor *backward*-equivalent.

In our implementation, the reachability fixed points are computed using a table that maps each location of $\mathcal{M}\|\mathcal{N}$ to a clock federation. Testing the forward or backward equivalence of two locations m_1 and m_2 of $\mathcal{M}$ then reduces to checking the equivalence of the entries for (m_1, n) and (m_2, n) for all locations n of $\mathcal{N}$.

A certificate based on both the forward and the backward equivalence can be obtained by computing the two equivalences in sequence, for example by first computing a forward and then a backward quotient: $\mathcal{C}_{FB} = (\mathcal{M}/_{\simeq_F})/_{\simeq_B}$.

5 Forward-Backward Reachability

The forward and backward equivalences introduced in the previous section base the equivalence either on forward reachability or on backward reachability, but not on both directions at the same time. This results in unnecessarily large quotients, as the following example illustrates.

Figure 3 shows the network $\mathcal{M}\|\mathcal{N}$. Locations b and c of the timed automaton $\mathcal{M}$ can safely be merged, because the final location remains unreachable in the network. However, b and c are not forward-equivalent, because they are forward-reachable at different times: location b is reached for $x > 3$, location c for $x > 4$ (both in conjunction with location f of $\mathcal{N}$). Since their backward reachability differs also ($x < 1$ for b and $x < 2$ for c), they are not backward-equivalent either.

In this section, we define a coarser equivalence that takes both forward and backward reachability into account. For a timed automaton $\mathcal{A}$ with finite semantics $\mathsf{sem}(\mathcal{A}) = (S, I, \Sigma, \Delta, \chi, F)$, we denote the *forward-backward reachable states*, that is, the states of $\mathsf{sem}(\mathcal{A})$ that are reachable from an initial state, and from which a final state is reachable, with $\mathsf{fbr}(\mathcal{A}) \subseteq S$ ($s \in \mathsf{fbr}(\mathcal{A}) \Leftrightarrow \exists i \in I.\ i \longrightarrow^* s \wedge \exists f \in F.\ s \longrightarrow^* f$).

Note that it is not sound to simply restrict the definitions of forward and backward equivalence to the forward-backward reachable states in $\mathcal{M}\|\mathcal{N}$. Consider a modification of the example from Figure 3, where the guard on the transition from location a to location b is changed to $x > 1$. The subsets of the forward-reachable states in b and c that are also backward reachable are both still empty. However, merging b and c is no longer safe, because the quotient would, for example, include the path to the final location that passes the merged location $\{b, c\}$ at time 1.5. In the following definition we therefore pose a slightly stronger requirement, by considering the forward-backward reachable states of $\mathcal{N}$ in isolation, rather than in the combination $\mathcal{M}\|\mathcal{N}$.

With $\mathsf{fbr}(\mathcal{A})_{\chi'}$ we denote the generalization of the regions to additional clocks with unconstrained value, that is, the integer part of the values for the new clocks as well as the relative order of the fractional part between the new and old clocks, among the new clocks, and compared to 0 is unconstrained. (We assume without loss of generality that the clock ceiling c_{max} is the same for all timed automata under consideration.) Let $L_{\mathcal{M}}$ and $\chi_{\mathcal{M}}$ be the locations and clocks, respectively, of $\mathcal{M}$, and let $\mathsf{sem}(\mathcal{M}\|\mathcal{N}) = (L_{\mathcal{M}} \times L_{\mathcal{N}} \times \mathcal{R}/_{\simeq_R}, I, \Sigma, \Delta, \chi, F)$. For locations $m_1, m_2 \in L_{\mathcal{M}}$ we define

$$
\begin{aligned}
m_1 \sim_F m_2 \Leftrightarrow \forall (n, \vec{t}) \in \mathsf{fbr}(\mathcal{N})_{\chi_{\mathcal{M}}}.\ & \\
\exists i_1 \in I \text{ s.t. } i_1 \longrightarrow^* \left[(m_1, n, \vec{t})\right]_R & \\
\Leftrightarrow \exists i_2 \in I \text{ s.t. } i_2 \longrightarrow^* \left[(m_2, n, \vec{t})\right]_R; & \\
m_1 \sim_B m_2 \Leftrightarrow \forall (n, \vec{t}) \in \mathsf{fbr}(\mathcal{N})_{\chi_{\mathcal{M}}}.\ & \\
\exists f_1 \in F \text{ s.t. } \left[(m_1, n, \vec{t})\right]_R \longrightarrow^* f_1 & \\
\Leftrightarrow \exists f_2 \in F \text{ s.t. } \left[(m_2, n, \vec{t})\right]_R \longrightarrow^* f_2. &
\end{aligned}
$$

We call two locations $m_1, m_2 \in L_{\mathcal{M}}$ *weakly forward-equivalent* if $m_1 \sim_F m_2$ and *weakly backward-equivalent* if $m_1 \sim_B m_2$. Compared to forward and backward equivalence as defined in the previous section, the requirements have been weakened in the sense that we ignore global states whose $\mathcal{N}$ part is incompatible with $\mathcal{N}$ alone. Ignoring these states is safe:

Theorem 5.1 *For a timed automaton $\mathcal{M}$ in a network $\mathcal{M}\|\mathcal{N}$, both $\mathcal{M}/_{\sim_F}$ and $\mathcal{M}/_{\sim_B}$ are certificates of $\mathcal{M}$.*

Proof: The proof of Theorem 4.1 applies directly. The only point that deserves more attention is the induction step for the *"only if"* direction: Here we have to argue why the restriction to the forward-backward

reachable fragment of $\mathcal{N}$ is sound. The soundness follows from the simple fact that the projection $s_0^e \to s_1^e \to \ldots \to s_n^e$ of a path $s_0 \to s_1 \to \ldots \to s_n$ of $\mathsf{sem}(\mathcal{M}/_{\sim_F} \| \mathcal{N})$ or $\mathsf{sem}(\mathcal{M}/_{\sim_B} \| \mathcal{N})$, respectively, is a is a path of $\mathsf{sem}(\mathcal{N})$. Thus, states of $\mathsf{sem}(\mathcal{N})$ that are ignored in the construction of $\sim_F$ or $\sim_B$ cannot be part of a state of $\mathsf{sem}(\mathcal{M}/_{\sim_F} \| \mathcal{N})$ or $\mathsf{sem}(\mathcal{M}/_{\sim_B} \| \mathcal{N})$, respectively, that occurs on a path from an initial to a final state in $\mathsf{sem}(\mathcal{M}/_{\sim_F} \| \mathcal{N})$ or $\mathsf{sem}(\mathcal{M}/_{\sim_B} \| \mathcal{N})$, respectively. □

A simple corollary of the theorem is that every equivalence relation $\sim$ that is finer than $\sim_F$ or $\sim_B$ can be used to obtain a certificate.

Corollary 5.2 *For a timed automaton $\mathcal{M}$ in a network $\mathcal{M} \| \mathcal{N}$ and an equivalence relation $\sim$ such that $\sim_F \supseteq \sim$ or $\sim_B \supseteq \sim$, $\mathcal{M}/_\sim$ is a certificate of $\mathcal{M}$.* □

6 Iterative Construction

A direct construction of the forward and backward reachable states is often too expensive. In this section, we therefore propose an approximative technique that over- and underapproximates these sets based on an over- and underapproximation of the successor operator. Constructing the approximations is cheap, but results in certificates, because the resulting equivalence relation is finer than the corresponding (weak) forward or backward equivalence.

In the second subsection, we show that the approximation can be stepwise refined, converging to the precise forward and backward reachable sets. The refinement steps are inexpensive, and intermediate results can be used to build intermediate certificates.

6.1 Approximating Reachability

The approximative reachability analysis is based on an abstraction structure, which we define to be any partition Π of the states S of the finite semantics $\mathsf{sem}(\mathcal{M} \| \mathcal{N}) = (S, I, \Sigma, \Delta, X, F)$ of $\mathcal{M} \| \mathcal{N}$. Intuitively, the state sets in Π constitute blocks in the state space that are either added completely or not at all by the approximative Succ and Pred operators. We obtain two versions of each operator: $\overline{\mathsf{Succ}}(P)$ computes the union of all state sets $P' \in \Pi$ of the partition Π such that *some* state in P' has a predecessor in P; $\underline{\mathsf{Succ}}(P)$ computes the union of all state sets P' in Π such that *all* states in P' have a predecessor in P. $\overline{\mathsf{Pred}}$ and $\underline{\mathsf{Pred}}$ are defined analogously:

$$\begin{array}{rcl}
\overline{\mathsf{Succ}}(P) & = & \bigcup\{P' \in \Pi \mid \exists s' \in P' \, \exists s \in P. \, s \to s'\}, \\
\underline{\mathsf{Succ}}(P) & = & \bigcup\{P' \in \Pi \mid \forall s' \in P' \, \exists s \in P. \, s \to s'\}, \\
\overline{\mathsf{Pred}}(P') & = & \bigcup\{P \in \Pi \mid \exists s \in P \, \exists s' \in P'. \, s \to s'\}, \\
\underline{\mathsf{Pred}}(P') & = & \bigcup\{P \in \Pi \mid \forall s \in P \, \exists s' \in P'. \, s \to s'\}.
\end{array}$$

Replacing the precise Succ and Pred operators in the fixed point construction from Section 4, we obtain four state sets: an overapproximation $\overline{\mathsf{FR}}$ and an underapproximation $\underline{\mathsf{FR}}$ of the forward reachable states, and, likewise, an overapproximation $\overline{\mathsf{BR}}$ and an underapproximation $\underline{\mathsf{BR}}$ of the backward reachable states.

$$\begin{array}{ll|ll}
\underline{\mathsf{FR}}_0 & = I & \underline{\mathsf{BR}}_0 & = F \\
\underline{\mathsf{FR}}_{i+1} & = \underline{\mathsf{FR}}_i \cup \underline{\mathsf{Succ}}(\underline{\mathsf{FR}}_i) & \underline{\mathsf{BR}}_{i+1} & = \underline{\mathsf{BR}}_i \cup \underline{\mathsf{Pred}}(\underline{\mathsf{BR}}_i) \\
\underline{\mathsf{FR}} & = \lim_i \underline{\mathsf{FR}}_i & \underline{\mathsf{BR}} & = \lim_i \underline{\mathsf{BR}}_i \\
\hline
\overline{\mathsf{FR}}_0 & = I & \overline{\mathsf{BR}}_0 & = F \\
\overline{\mathsf{FR}}_{i+1} & = \overline{\mathsf{FR}}_i \cup \overline{\mathsf{Succ}}(\overline{\mathsf{FR}}_i) & \overline{\mathsf{BR}}_{i+1} & = \overline{\mathsf{BR}}_i \cup \overline{\mathsf{Pred}}(\overline{\mathsf{BR}}_i) \\
\overline{\mathsf{FR}} & = \lim_i \overline{\mathsf{FR}}_i & \overline{\mathsf{BR}} & = \lim_i \overline{\mathsf{BR}}_i
\end{array}$$

Our implementation again computes the four approximated reachability fixed points using a table that maps each location of $\mathcal{M} \| \mathcal{N}$ to a clock federation.We can establish the forward/backward equivalence of two locations m_1 and m_2 of $\mathcal{M}$ once the entries for the over- and underapproximation coincide for the entries for (m_1, n) and (m_2, n) for all locations n of $\mathcal{N}$ by the same technique as in the precise method described in Section 4. Likewise, we can exclude the forward/backward equivalence of two locations m_1 and m_2 as soon as, for some location n of $\mathcal{N}$, the underapproximation of the set of regions attached to (m_1, n) is not a subset of the overapproximation of the regions attached to (m_2, n). We can approximate weak forward and weak backward equivalence by intersecting with $\mathsf{fbr}(\mathcal{N})_{\chi_M}$ as described in Section 5.

6.2 Abstraction Refinement

Finer abstraction structures result in coarser equivalence relations and, hence, smaller quotients. We iteratively construct a sequence of successively coarser equivalences by stepwise refining the partition Π^i. This results in an ascending chain

$$\underline{\mathsf{FR}}^0 \subseteq \underline{\mathsf{FR}}^1 \subseteq \underline{\mathsf{FR}}^2 \ldots$$

of underapproximations and a descending chain

$$\overline{\mathsf{FR}}^0 \supseteq \overline{\mathsf{FR}}^1 \supseteq \overline{\mathsf{FR}}^2 \supseteq \ldots$$

of overapproximations. Both chains converge to FR when Π^i converges to the set of singleton sets. (In

	Certificate synthesis					UPPAAL			
						explored states		time [ms]	
Benchmark	$\mathcal{N}$	$\mathcal{M}$	$\mathcal{C}$	factor	time [ms]	$\mathcal{M}\|\mathcal{N}$	$\mathcal{C}\|\mathcal{N}$	$\mathcal{M}\|\mathcal{N}$	$\mathcal{C}\|\mathcal{N}$
FP 2	43	7	5	1.4	4	47	35	8	7
FP 3	64	49	17	2.9	21	273	153	10	13
FP 4	85	343	64	5.4	339	1471	1116	19	41
FP 5	106	2401	218	11	6702	7493	9527	73	330
CP 2	11	5	2	2.5	3	17	9	6	6
CP 3	19	25	4	6.3	9	65	33	8	7
CP 4	35	125	8	15.6	14	257	182	10	10
CP 5	67	625	16	39.1	68	1025	321	22	18
CP 6	131	3125	32	97.7	394	4097	969	77	41
CP 7	259	15625	64	244	2110	16835	2913	339	109
CP 8	515	78125	128	610	11961	65537	16982	1675	584
GPS 2	17	6	3	2	2	48	12	6	6
GPS 3	17	36	4	9	6	201	61	8	8
GPS 4	17	216	6	36	44	831	27	12	10
GPS 5	17	1296	7	185	307	3405	33	30	14
GPS 6	17	7776	8	972	3020	13863	36	121	18
GPS 7	17	46656	9	5184	22239	56181	42	613	16
GPS 8	17	279936	10	27994	381624	226911	51	3041	19

Table 1. Certificate synthesis for Fischer's protocol (FP), the Combination Platform (CP), and the Gear Production Stack (GPS). The table shows the size of the environment automaton ($\mathcal{N}$), the size of the component automaton ($\mathcal{M}$), the size and reduction factor of the certificate ($\mathcal{C}$, factor), the running time of the direct certificate construction, and the performance of UPPAAL on $\mathcal{M}\|\mathcal{N}$ and $\mathcal{C}\|\mathcal{N}$, given as the number of states explored and the running time. All benchmarks were measured on an AMD Opteron processor with 2.6 GHz.

practice, the precise equivalence is usually found much earlier.)

We start with some initial partition Π^0 that separates the final states from all other states. In our implementation, Π^0 partitions the states according to their locations.

In every iteration of the refinement loop, we split Π^i with a set $\hat{P} \subseteq S$ of states, resulting in the new partition

$$\Pi^{i+1} \;=\; \bigcup \big\{ \{P_1, P_2\} \mid \exists P \in \Pi^i.\; P_1 = P \cap \hat{P} \;\wedge\; P_2 = P \setminus \hat{P} \big\}.$$

The set $\hat{P}$ is chosen by the refinement heuristic. A simple strategy that improves the approximation of the forward reachable states is to choose $\hat{P} = \mathsf{Succ}(\underline{\mathsf{FR}}^i)$.

In each new iteration, some results of the previous iteration can be reused. For example, the inclusion $\underline{\mathsf{FR}}^i \subseteq \underline{\mathsf{FR}}^{i+1}$ suggests to use $\underline{\mathsf{FR}}_0^{i+1} = \underline{\mathsf{FR}}^i$ (instead of $\underline{\mathsf{FR}}_0^{i+1} = I$). Note that this guarantees that no element of any partition is added twice to the underapproximation. Likewise, the inclusion $\underline{\mathsf{FR}}^i \subseteq \mathsf{FR} \subseteq \overline{\mathsf{FR}}^i$ suggests to use $\overline{\mathsf{FR}}_0^i = \underline{\mathsf{FR}}^i$. An iterative computation of the backward-reachable states can be defined analogously.

After every iteration, the approximation defines an equivalence $\approx$, which can be used to compute an *intermediate certificate* $\mathcal{C}'$. While $\approx$ is finer than the precise forward or backward equivalences, it often reduces $\mathcal{M}$ significantly already after a few refinement steps. In that case, we can replace $\mathcal{M}$ with $\mathcal{C}'$, and, if desired, switch between computing the forward and backward equivalence. A reuse of intermediate results is still possible after replacing $\mathcal{M}$ with $\mathcal{C}'$: we again start with $\underline{\mathsf{FR}}_0^{i+1} = \underline{\mathsf{FR}}^i/_{\approx}$ and $\underline{\mathsf{BR}}_0^{i+1} = \underline{\mathsf{BR}}^i/_{\approx}$.

7 Benchmarks and Results

Table 1 shows experimental results with our prototype implementation on Fischer's mutual exclusion protocol (FP 2–5) and two case studies provided by our industrial partners BPS IT-Solutions and META-LEVEL Software AG (CP 2–8 and GPS 2–8). For each benchmark, the table shows the size of the environment automaton and the component automaton, the size and reduction factor of the certificate, the running time of the certificate synthesis, and the performance of the UPPAAL [18] model checker on $\mathcal{M}\|\mathcal{N}$ and $\mathcal{C}\|\mathcal{N}$.

Benchmark	$\mathcal{M}$	Iteration 1		Iteration 10		Iteration 15	
		$\mathcal{C}$	time [ms]	$\mathcal{C}$	time [ms]	$\mathcal{C}$	time [ms]
FP 2	7	7	5	—	—	—	—
FP 3	49	49	6	46	43	—	—
FP 4	343	343	47	343	1087	129	2743
FP 5	2401	2377	581	2377	33174	1914	102829
CP 2	5	5	3	—	—	—	—
CP 3	25	18	4	6	6	—	—
CP 4	125	66	9	26	48	10	72
CP 5	625	258	27	157	253	22	675
CP 6	3125	1026	143	810	2171	149	9659
CP 7	15625	4098	879	3685	20450	1256	259731
CP 8	78125	16386	5217	15658	409907	8258	8144794
GPS 2	6	6	4	5	9	4	12
GPS 3	36	31	10	31	18	31	38
GPS 4	216	139	15	139	19	139	44
GPS 5	1296	607	52	607	153	607	320
GPS 6	7776	2587	310	2587	641	2587	1541
GPS 7	46656	10831	2043	10831	3376	10831	7179
GPS 8	279936	44779	14171	44779	20175	44779	35973

Table 2. Incremental construction of the certificate for Fischer's protocol (FP), the Combination Platform (CP), and the Gear Production Stack (GPS). The table shows the size of the component automaton ($\mathcal{M}$), and the size of the certificate ($\mathcal{C}$), and the running time of the synthesis algorithm after 1, 10, and 15 iterations. All benchmarks were measured on an AMD Opteron processor with 2.6 GHz.

Fischer's protocol (FP). Fischer's mutual exclusion protocol is a standard benchmark for the verification of real-time systems, parameterized in the number of participants.

Combination Platform (CP). The combination platform benchmark, provided by META-LEVEL Software AG, models a platform used in car manufacturing that combines several testing machines with units for cleaning and polishing. The different units work in parallel and synchronize after completing their tasks. The critical property of the combination platform is that the work is completed by a certain deadline. The benchmark is parameterized in the number of sub-controllers included in the model.

Gear Production Stack (GPS). The gear production stack benchmark, provided by BPS IT-Solutions, models a production machine for gear wheels, which consists of units for casting, hardening, and polishing. The units work sequentially on a single workpiece. Like in the CP benchmark, the property requires that the work is completed by a certain deadline. The benchmark is again parameterized in the number of sub-controllers included in the model.

Comparing the growth of the component $\mathcal{M}$ with the growth of the certificate $\mathcal{C}$ in our parameterized benchmarks, it is evident that the certificate grows much slower. The difference is most clear-cut in the GPS benchmarks, where the size of $\mathcal{M}$ grows exponentially, while the size of $\mathcal{C}$ only grows linearly.

The running times for the certificate synthesis shown in Table 1 refer to the direct construction of the certificate. Our implementation approximates the weak equivalence relations defined in Section 5 based on a cheap approximation of the set $\mathsf{fbr}(\mathcal{N})$ that partitions the states according to their locations.

Table 2 shows experimental results for the incremental certificate synthesis from Section 6. Our implementation starts with an initial abstraction that partitions the states according to their location. As a result, the first intermediate certificate already shows a significant reduction in the number of locations.

If the incremental algorithm is run until termination, the accumulated running time is, in our experiments, higher than the running time of the direct construction. One reason for this effect may be our simple refinement heuristic ($\hat{P} = \mathsf{Succ}(\underline{\mathsf{FR}}^i)$). Good heuristics for choosing the refinement sets, as well as the initial abstraction and the termination point of the iteration, are interesting topics for future research.

8 Conclusions

We have presented a solution to the problem of synthesizing a certificate for a timed automaton $\mathcal{M}$ in a network $\mathcal{M}\|\mathcal{N}$. In contrast to the NP hardness of finding the minimal certificate, the cost of our construction is just linear in the number of locations; nevertheless, the dramatic decrease in size from the component $\mathcal{M}$ to the certificate $\mathcal{C}$ in our experimental results suggests that the certificates found by our construction are close to minimal.

Since our approach is based on a reachability construction, the worst-case complexity in terms of the number of clocks is exponential. To address this issue we have proposed an iterative approximation method that can be interrupted at any time to produce a sound certificate.

We believe that the certificates constructed by *certifying* model checkers will be useful to designers in understanding which component requirements are *hard* in the sense that they are necessary to guarantee the safety of the system, and which requirements are *soft*, that is, relevant for the quality provided by the system but not for its safety. Such a classification is an important piece of documentation and useful in future adaptations of the verified design.

References

[1] R. Alur, P. Cerny, P. Madhusudan, and W. Nam. Synthesis of interface specifications for Java classes. In *Proceedings of the 32nd Annual Symposium on Principles of Programming Languages (POPL 2005), 12–14 January, Long Beach, California, USA*, pages 98–109. ACM Press, 2005.

[2] R. Alur, C. Courcoubetis, N. Halbwachs, D. L. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *Proceedings of the Third International Conference on Concurrency Theory (CONCUR 1992), 24–27 August, Stony Brook, NY, USA*, volume 630 of *Lecture Notes in Computer Science*, pages 340–354. Springer-Verlag, 1992.

[3] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] R. Alur, P. Madhusudan, and W. Nam. Symbolic compositional verification by learning assumptions. In *Proceedings of the 17th International Conference on Computer Aided Verification (CAV 2005), 6–10 July, Edinburgh, Scotland, UK*, volume 3576 of *Lecture Notes in Computer Science*, pages 548–562. Springer-Verlag, 2005.

[5] F. Balarin. Approximate reachability analysis of timed automata. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS 1996), 4–6 December, Washington, DC, USA*, pages 52–61. IEEE Computer Society, 1996.

[6] J. Bengtsson. *Clocks, DBM, and States in Timed Systems*. PhD thesis, Uppsala University, 2002.

[7] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR 1998), 8–11 September, Nice, France*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500. Springer-Verlag, 1998.

[8] J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning assumptions for compositional verification. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003), 7–11 April, Warsaw, Poland*, volume 2619 of *Lecture Notes in Computer Science*, pages 331–346. Springer-Verlag, 2003.

[9] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *Proceedings of the 4th Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 1998), 28 March–4 April, Lisbon, Portugal*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer-Verlag, 1998.

[10] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS 1996), 4–6 December, Washington, DC, USA*, pages 73–81. IEEE Computer Society, 1996.

[11] H. Dierks, S. Kupferschmid, and K. G. Larsen. Automatic abstraction refinement for timed automata. In *Proceedings of the 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2007), 3–5 October, Salzburg, Austria*, volume 4763 of *Lecture Notes in Computer Science*, pages 114–129. Springer-Verlag, 2007.

[12] B. Finkbeiner, H.-J. Peter, and S. Schewe. RESY: Requirement synthesis for compositional model checking. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008), 31 March–3 April, Budapest, Hungary*, volume 4963 of *Lecture Notes in Computer Science*, pages 463–466. Springer-Verlag, 2008.

[13] B. Finkbeiner, S. Schewe, and M. Brill. Automatic synthesis of assumptions for compositional model checking. In *Proceedings of the 26th International Conference on Formal Methods for Networked and Distributed Systems (FORTE 2006), 26–29 September, Paris, France*, volume 4229 of *Lecture Notes in Computer Science*, pages 143–158. Springer Verlag, 2006.

[14] G. Holzmann. *The Spin Model Checker, Primer and Reference Manual*. Addison-Wesley, Reading, Massachusetts, 2003.

[15] I. Kang, I. Lee, and Y.-S. Kim. An efficient state space generation for the analysis of real-time systems. *IEEE Transactions on Software Engineering*, 26(5):453–477, 2000.

[16] F. Laroussinie and K. G. Larsen. Compositional model checking of real time systems. In *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR 1992), 21–24 August, Philadelphia, PA, USA*, volume 962 of *Lecture Notes in Computer Science*, pages 27–41. Springer-Verlag, 1995.

[17] F. Laroussinie and K. G. Larsen. CMC: A tool for compositional model-checking of real-time systems. In *Proceedings of the 11th International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE 1998), 3–6 November, 1998, Paris, France*, volume 135 of *IFIP Conference Proceedings*, pages 439–456. Kluwer, 1998.

[18] K. Larsen, P. Petterson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1+2):134–152, Dec. 1997.

[19] K. G. Larsen. A context dependent equivalence between processes. *Theoretical Computer Science*, 49(2-3):185–215, 1987.

[20] K. S. Namjoshi. Certifying model checkers. In *Proceedings of the 13th International Conference on Computer Aided Verification (CAV 2001), 18–22 July, Paris, France*, volume 2102 of *Lecture Notes in Computer Science*, pages 2–13. Springer-Verlag, 2001.

[21] M. Yannakakis and D. Lee. An efficient algorithm for minimizing real-time transition systems. *Formal Methods in System Design*, 11(2):113–136, 1997.