From Scenarios to Timed Automata

Neda Saeedloei[1] and Feliks Kluźniak[2]

[1] Southern Illinois University, Carbondale *
neda@cs.siu.edu,
[2] Logic Blox
feliks.kluzniak@logicblox.com

Abstract. We describe a new method of synthesizing a formal model for real-time systems from scenarios. Scenarios, formally defined as *Timed Event Sequences*, together with *mode graphs* are used to describe behaviors of real-time systems. Given a set of Timed Event Sequences and a mode graph, our synthesis method constructs a *minimal*, *acyclic*, timed automaton that models the specified aspects of the system. We formalize criteria that a set of scenarios must satisfy in order to make it feasible to generate such an automaton.

Keywords: Formal models; scenarios; timed automata.

1 Introduction

Model-based design has been used as an effective approach to the process of designing, analysis, and verification of complex systems. The process can greatly benefit from building a formal model, i.e., one that is expressed in a formal language with well-defined semantics. An important advantage of modeling is that one obtains insight into how a physical realisation of the system would behave in the real world.

A formal model can be used as a high-level "prototype": it can be experimented with and iteratively improved. (The cost of doing so is significantly lower than that of experimenting with and improving a real implementation of the system.) Depending on how the model is constructed, such experimentation usually takes the form of either formally deriving logical conclusions, or of automatic simulation. The overall purpose is to obtain a validated model, i.e., one whose "behavior" does indeed comply with the desired behavior of the modeled system. The insight mentioned in the preceding paragraph arises out of the necessity of formulating very clear criteria for (or examples of) desirable and undesirable behaviors.

While the use of model-based design is promising and rapidly growing, a major problem is the lack of good formal requirements, which are the starting point in building a model. System requirements are often incomplete, ambiguous or

* The preliminary research was carried out while the first author was at the University of Minnesota, Duluth.

very low level. Incomplete requirements cannot be used for building realistic systems, as some parameters and functionalities are missing and must be guessed. Real-time systems, in particular, are often safety-critical systems, and unspecified behaviors and missing cases cannot be tolerated. Ambiguous requirements must be made more precise before the formal model is complete (and the act of analysing and attempting to model ambiguous requirements can be very helpful in detecting the ambiguities). Finally, very low level requirements are difficult to understand and reason about, so tend to be less useful for modeling purposes. It is exactly the unavailability of satisfactory requirements specifications that makes modeling such an important step in the process of constructing a system.

In this paper we focus on two important questions: (1) how to express requirements, and (2) how to obtain formal models of real-time systems from requirements. Our formal models will take the form of timed automata [1].

We propose a new method for synthesizing a timed automaton model of a real-time system from *scenarios*. A scenario can be viewed as a description of a set of *partial* behaviors of a real-time system during a time interval: it describes not only the events that occur in the system, but also the timing relations among the events. All behaviors "allowed" by a particular scenario must satisfy the same time constraints.

The events in a scenario include both the system's internal events and its interactions with users or the environment. A set of scenarios can capture the important aspects of the required system behaviors, and be the starting point for constructing a formal model of the system.

We introduce *Timed Event Sequences (TES)*[1] to describe scenarios formally, precisely, and at a high level of abstraction. We also use *mode graphs* (others have called them "mode diagrams" [8]) to specify the events that are possible at various points in the history of the system. Our scenarios, together with mode graphs, can be viewed as high level descriptions of real-time systems. As such, they are easy to understand and use by practitioners who utilise scenarios to elicit system requirements. The other alternative would be to build timed automata directly. But people who provide information about the requirements will not always be comfortable with timed automata: for some of them mode graphs and scenarios might be easier to handle (this observation seems to have motivated different work by others [8]).

Our method constructs a timed automaton from a set of TES and a mode graph. The constructed automaton is an *acyclic*[2] automaton with a minimum number of locations, which captures all those behaviors of the system that are described by the set of TES. All the possible runs of the constructed timed automaton are possible behaviors of the system in the sense of being runs of the mode graph. The principal value of the constructed automaton is that it caters for the time constraints introduced in the scenarios.

[1] We will use the abbreviation for both the singular and the plural form of the term.
[2] The acyclicity of our automaton does not prevent it from being used as a timed automaton with infinite runs, as will be explained in Section 6.

Our goal of synthesizing timed automata with a minimal number of locations is motivated by the fact that the "state explosion problem" has been a major challenge in model checking concurrent systems [5].

We construct *acyclic* timed automata, because the conventional formalism of timed automata does not allow inclusion of variables other than clocks. By allowing cycles the formalism would have to be extended in order to express a limit on the number of iterations (e.g., the number of attempts to enter a PIN).

From a bird's eye perspective, our approach can be described as follows:

1. A mode graph $\mathcal{M}$ defines the universe $B_{\mathcal{M}}$ of all the possible behaviors of the system: these are, roughly, the runs of $\mathcal{M}$ when treated as a finite automaton.
2. A scenario A imposes timing constraints on certain kinds of concrete behaviors, thus defining $\mathit{Allowed}(A) \subset B_{\mathcal{M}}$: the set of those possible behaviors that are allowed by A.
3. Two scenarios, A and B, together define the set of allowed behaviors as $\mathit{Allowed}(A) \cap \mathit{Allowed}(B)$.
4. The constructed timed automaton exhibits behaviors that are allowed by all the scenarios and are described by some of them. If the set of exhibited behaviors is smaller than expected, then either the constraints in some scenarios are too strict, or certain kinds of behaviors have not yet been specified. The activity of specifying a system must of necessity be iterative: that is why the automaton must be constructed automatically.

2 Timed automata

We now present a brief overview of timed automata [1].

For a set C of clock variables, the set $\Phi(C)$ includes *clock constraints* 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 *timed automaton* is a tuple $\mathcal{A} = \langle E, Q, Q_0, Q_f, C, R \rangle$, where

- E is a finite alphabet;
- Q is the (*finite*) set of locations;
- $Q_0 \subset Q$ is the set of initial locations;
- $Q_f \subseteq Q$ is the set of final locations;
- C is a finite set of *clock* variables;[3]
- $R \subseteq Q \times Q \times E \times 2^C \times 2^{\Phi(C)}$ is the set of transitions of the form $(q, q', e, \lambda, \phi)$, where the set $\lambda \subseteq C$ is the set of clocks to be reset with this transition, and ϕ is a set of clock constraints over C.

A time sequence $\tau = \tau_1 \tau_2 ...$ is an infinite sequence of (time) values $\tau_i \in \mathbb{R}^{\geq 0}$, satisfying two requirements:

- *Monotonicity*: τ increases strictly monotonically, i.e., $\tau_i < \tau_{i+1}$ for all $i \geq 1$.
- *Progress*: For every $t \in \mathbb{R}^{\geq 0}$, there is some $i \geq 1$ such that $\tau_i > t$.

A *timed word* over an alphabet E is a pair (σ, τ) where $\sigma = \sigma_1 \sigma_2 ...$ is an infinite word over E and τ is a time sequence.

[3] We will follow the usual convention and use "clocks" instead of "clock variables".

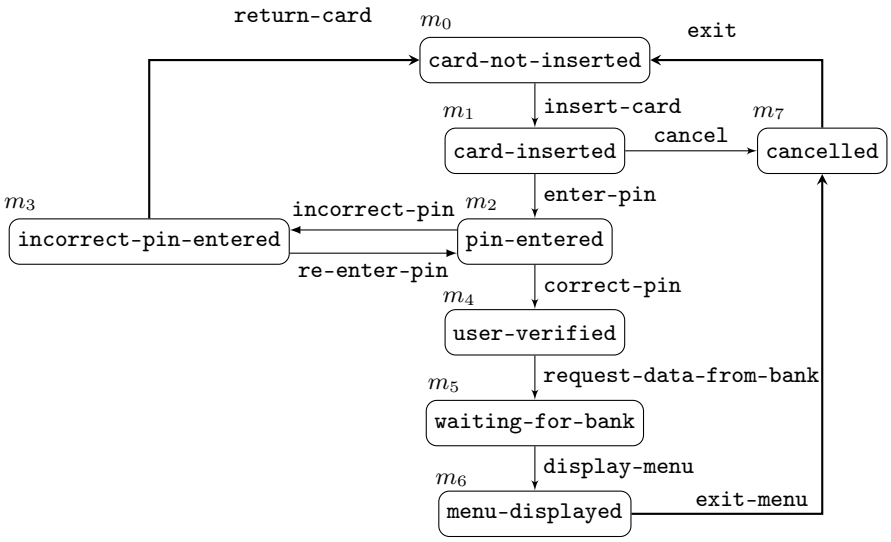

Fig. 1. A mode graph corresponding to the ATM

A *clock interpretation* for a set C of clocks assigns a value in $\mathbb{R}^{\geq 0}$ to each clock; that is, it is a mapping from C to $\mathbb{R}^{\geq 0}$. We say that a clock interpretation ν for C satisfies a set of clock constraints ϕ over C iff every clock constraint in ϕ evaluates to true after replacing each clock variable c with $\nu(c)$.

For $\tau \in \mathbb{R}, \nu + \tau$ denotes the clock interpretation which maps every clock c to the value $\nu(c) + \tau$. For $Y \subseteq C$, $[Y \mapsto \tau]\nu$ denotes the clock interpretation for C which assigns τ to each $c \in Y$, and agrees with ν over the rest of the clocks.

A run ρ of $\mathcal{A}$ over a timed word (σ, τ) is an infinite sequence of the form

$$\rho : \langle q_0, \nu_0 \rangle \xrightarrow[\tau_1]{\sigma_1} \langle q_1, \nu_1 \rangle \xrightarrow[\tau_2]{\sigma_2} \langle q_2, \nu_2 \rangle \xrightarrow[\tau_3]{\sigma_3} \ldots$$

with $q_i \in Q$ and $\nu_i \in [C \mapsto \mathbb{R}^{\geq 0}]$, for all $i \geq 0$, satisfying two requirements:

- $q_0 \in Q_0$, and $\nu_0(c) = 0$ for all clocks $c \in C$;
- for every $i \geq 1$ there is a transition in R 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 $[\lambda_i \mapsto 0](\nu_{i-1} + \tau_i - \tau_{i-1})$.

3 Specifying Scenarios

We will now describe our representation of scenarios. We will use the initial behavior of an Automatic Teller Machine (ATM) to illustrate some terms and definitions.

Intuitively, a scenario describes a set of partial behaviors of a system during a time interval. We use mode graphs to describe *all* the possible behaviors of a system. A mode graph (cf. [8]) is a deterministic state machine that describes the high level behavior of a system. Modes can be viewed as visible states of

a system. A transition is triggered by an event allowable in the current mode and brings the system to a new mode. For example, Fig. 1 shows the mode graph that outlines the initial behavior of the ATM controller and a potential customer (user). The transition from *pin-entered* to *user-verified* is triggered by a *correct-pin* event.

Scenarios are used to put various constraints on certain behaviors of a system. Examples of constraints include time constraints on certain events or constraints on the number of occurences of some events, e.g., a limit on the number of times a PIN can be re-entered.

We use Timed Event Sequences to formally represent scenarios. A Timed Event Sequence (formally defined in Sec. 4) contains:

- The initial and the final mode of the specified scenario.
- A sequence of timed events. We assume each event occurs at some time w, shown by the *wall clock* when the event occurs. An event may be augmented with a set of time annotations that impose restrictions on relations between the time of the event and the times of some earlier events in the scenario.

Fig. 2 shows two scenarios, which specify the ATM's initial behavior. Scenario 1 describes behaviors in which the user enters an incorrect PIN, and then the correct PIN in the second attempt. It includes a sequence of seven timed events, along with the initial and the final modes. Initially the mode of the system is m_0 (*card-not-inserted*). After inserting the card at time t_0, the user enters the PIN at time $w > t_0$, such that the time difference between the two events, i.e., $w - t_0$ is within $[5, 60]$ seconds. Upon receiving the PIN, the ATM notifies the user that the PIN was incorrect. The user then enters another PIN within $[5, 60]$ seconds since inserting the card. Once the user is verified (mode m_4), the system displays the menu within the next 5 seconds. The new mode at this point is m_6 (*menu-displayed*). Please note that t_j, where $0 \leq j \leq 7$, is used to denote the time of leaving mode m_j (most recently).

Scenario 2 describes behaviors in which the user enters the correct PIN on the first try. The time annotations are a subset of those in scenario 1.

It is easy to see that each of these scenarios allows an infinite number of slightly different behaviors: a scenario directly captures the salient features of a class of allowable behaviors.

4 A Formal Description of Mode Graphs and TES

Mode Graphs. A *mode graph* is a tuple $\mathcal{M} = (M, m_0, m_f, \Sigma, T)$, where M is a finite set of modes, m_0 is the initial mode, m_f is the final mode (which can be identical to m_0), Σ is a set of events, and $T : M \times \Sigma \to M$ is a transition function. The latter will be represented as a set of transitions, i.e., triples of the form (m_i, e, m_j), where m_i and m_j, are modes in M, and e is an event in Σ. We assume events are unique in the mode graph: for an event $e \in \Sigma$, there is at most one transition in T that is labelled with e.[4]

[4] This is just a convenient convention, not a real restriction.

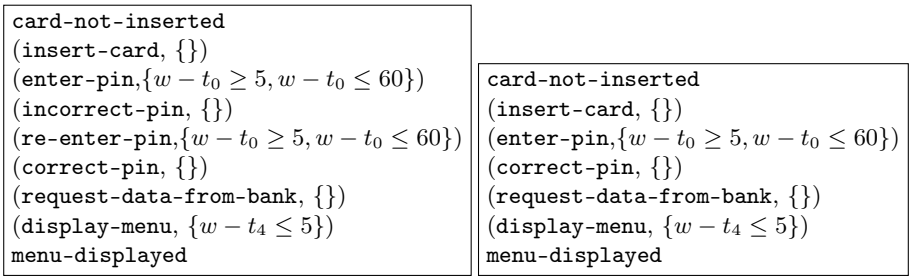

Fig. 2. Two scenarios showing two alternative sets of initial behaviors of the ATM

A *run* r of $\mathcal{M}$ is a (possibly infinite) sequence of the form $r : s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} s_3 \xrightarrow{e_3} \ldots$ (where $s_1 = m_0$ and $s_i \in M$), such that for every $i \geq 1$ there is a transition in T of the form (s_i, e_i, s_{i+1}). Any contiguous subsequence of a run is called a *partial run* (so a run is also a partial run).

We define $source(e) = m$, if $(m, e, n) \in T$, for any mode $n \in M$.

If m and n are modes in M, then m *dominates* n if and only if all paths from the initial mode to n pass through m [9]. We denote the *dominance relation* on M by $\succeq$: $m \succeq n$ iff m dominates n (we also say that n is dominated by m). We write $m \succ n$ to denote that $m \succeq n$ and $m \neq n$. ($\succeq$ is a partial order.)

We extend the definition of dominated modes to *dominated events*: an event e is *dominated* by mode m iff $m \succeq source(e)$. For example, in Fig. 1, *card-not-inserted, card-inserted*, *pin-entered*, *user-verified* and *waiting-for-bank* are all the dominating modes of *waiting-for-bank*. The event *display-menu* is dominated by all the modes that dominate *waiting-for-bank*.

Let $V = \{t_1, t_2, ..., t_n\}$ be a set of time variables, such that $|V| \geq |M|$.

To each mode m_i of M that appears in a scenario we assign $t_i \in V$, which is interpreted as the time of leaving m_i. If there is a cycle involving mode m_i, then t_i corresponds to the time of the *most recent* event occurrence at m_i *in that scenario* (since the most recent visit at m_0). For example, if a scenario for the mode graph of Fig. 1 describes an iteration between modes m_2 and m_3, then t_2 corresponds to the time of the most recent notification of an incorrect PIN in the scenario.

We use $\Phi(V)$ to denote the set of *time annotations* of the form $w - t_j \sim a$, where w is the time currently shown by the global wall clock, t_j is a time variable in V, $\sim \in \{\leq, \geq, <, >, =\}$, and a is a constant in the set of rational numbers.

Timed Event Sequences. Given a mode graph $\mathcal{M} = (M, m_0, m_f, \Sigma, T)$, a *Timed Event Sequence* ξ is a tuple of the form $\langle m^{initial}, \Psi, m^{final} \rangle$, where $m^{initial}$ and m^{final} are in M, and Ψ is a *non-empty* sequence of timed events of the form (e_k, ϕ_k), $1 \leq k \leq n$, where $e_k \in \Sigma$, and $\phi_k \in 2^{\Phi(V)}$ is a set of time annotations associated with e_k. $m^{initial}$ and m^{final} are the first and last modes of ξ.

We define $initial\_mode(\xi) = m^{initial}$ and $final\_mode(\xi) = m^{final}$. We also define $events(\xi)$ to be the sequence $e_1 e_2 \ldots e_n$ of events in ξ.

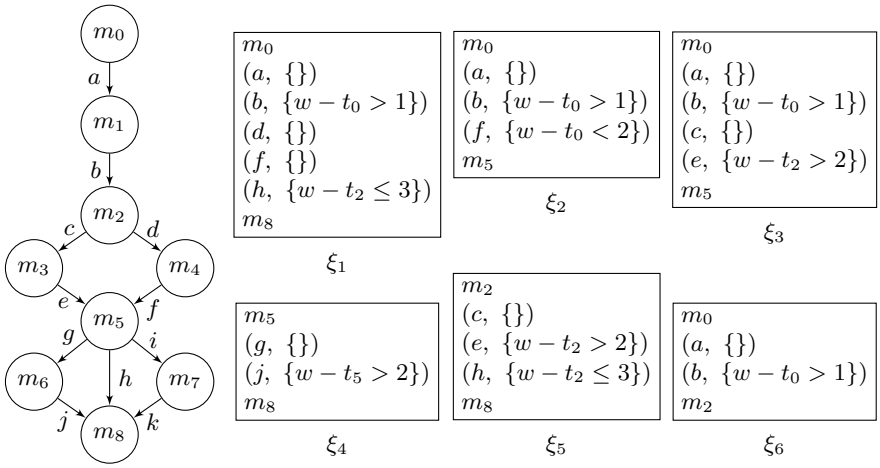

Fig. 3. A mode graph and a few TES

Compatibility with Mode Graph. A TES $\xi = \langle m^{initial}, \Psi, m^{final} \rangle$ is *compatible with* $\mathcal{M}$, if

- for $events(\xi) = e_1 \dots e_n$, there exists a partial run $s_1 \xrightarrow{e_1} s_2 \dots \xrightarrow{e_n} s_{n+1}$ of $\mathcal{M}$, such that $s_1 = m^{initial}$, $s_{n+1} = m^{final}$ and $s_i \in M$, for $1 \leq i \leq n$;
- for a timed event (e_k, ϕ_k) in Ψ and time annotation $w - t_a \sim b \in \phi_k$, t_a corresponds to the time of the most recent event e_j that occurred before e_k in ξ, such that $s_j = m_a$ and $m_a \succ s_k$.

The second requirement means that a time annotation accompanying an event e can refer *only* to the times of previous events which originated in modes that dominate e. This fundamental *dominance assumption* guarantees that all time variables are *well-defined* (i.e., each time variable used in a time annotation on a transition is being defined on every path that reaches the transition).

This notion is best illustrated by an example. In the mode graph of Fig. 3, each mode m_i is associated with time t_i. In scenario ξ_1, the time annotation on event h refers to the time of leaving m_2, which dominates h. Hence, t_2 is well-defined regardless of the path taken from m_0 to m_5.

If h were annotated with a reference to t_3, t_3 would not be well-defined, as m_3 does not dominate h: if m_5 were reached through m_4, t_3 would be undefined.

Observe that the TES ξ_2 of Fig. 3 is not compatible with the mode graph of Fig. 3, as the sequence *abf* does not correspond to a partial run of the mode graph. The rest of the TES in Fig. 3 are compatible with the mode graph.

Behaviors. Given a mode graph $\mathcal{M} = (M, m_0, m_f, \Sigma, T)$, we define a *behavior* as a finite timed word (e, w) over Σ, where $e = e_1 e_2 e_3 ... e_n$, such that there exists a run $m_0 \xrightarrow{e_1} s_2 \xrightarrow{e_2} s_3 \xrightarrow{e_3} \dots \xrightarrow{e_n} m_f$ of $\mathcal{M}$ (in which m_0 and m_f occur only once), and $w = w_1 w_2 w_3 \dots w_n$ is a monotonically increasing sequence of real numbers. Each w_i represents the value of the wall clock when e_i occurs. We use $B_{\mathcal{M}}$ to denote the set of all such behaviors.

A non-empty contiguous subsequence $(e_k...e_m, w_k...w_m)$ of a behavior $\mathcal{B} = (e, w)$ is *covered* by a TES ξ iff $e_k...e_m = events(\xi)$.

A behavior $\mathcal{B}$ is *relevant* to TES ξ iff it includes a subsequence that is covered by ξ. Otherwise, it is *irrelevant* to ξ.

All the behaviors in $\mathcal{B}_1 = \{(abceik, w_1w_2w_3w_4w_5w_6) \mid \forall_{1<i\leq 6}\ w_i > w_{i-1}\}$ are relevant to ξ_3 of Fig. 3, as their subsequence $(abce, w_1w_2w_3w_4)$ is covered by ξ_3. Similarly, the behaviors are relevant to ξ_6 on account of the subsequence (ab, w_1w_2). But the behaviors are irrelevant to ξ_4, for example.

A behavior $\mathcal{B}$ is *covered* by a set of TES, Ξ, iff $\mathcal{B}$ can be partitioned into a series of non-empty contiguous subsequences, such that each subsequence is covered by a *different* ξ in Ξ. (Different, to avoid unwanted loops.)

All the behaviors in $\mathcal{B}_2 = \{(abcegj, w_1w_2w_3w_4w_5w_6) \mid \forall_{1<i\leq 6}\ w_i > w_{i-1}\}$ are covered by the set $\{\xi_3, \xi_4\}$ of TES in Fig. 3.

We define $Covered(\Xi) = \{\mathcal{B} \in B_{\mathcal{M}} \mid \mathcal{B} \text{ is covered by } \Xi\}$.

A behavior $\mathcal{B}$=$(e_1e_2 \dots e_n, w_1w_2 \dots w_n)$ is *allowed* by a TES ξ iff, for every non-empty contiguous subsequence $(e_k \dots e_m, w_k \dots w_m)$ of $\mathcal{B}$ that is covered by ξ, the following holds: for every $k \leq i \leq m$, $w - t_j \sim a \in \phi_i$ in ξ evaluates to true when w is replaced by w_i and t_j by the time of leaving mode m_j (for the last time before e_i in this covered subsequence). The dominance assumption ensures that the latter will be well-defined.

Notice that a behavior that is irrelevant to ξ is allowed by ξ.

All the behaviors in $\mathcal{B}_3 = \{(abceh, w_1w_2w_3w_4w_5) \mid (\forall_{1<i\leq 5}\ w_i > w_{i-1}) \wedge w_2 - w_1 > 1 \wedge w_5 - w_3 \leq 3\}$ are allowed by ξ_3, ξ_5 and ξ_6 in Fig. 3. The behaviors in $\mathcal{B}_3$ are allowed also by ξ_4, as they are irrelevant to ξ_4.

The set of behaviors allowed by a TES ξ is denoted by $Allowed(\xi) \subset B_{\mathcal{M}}$.

With a slight abuse of notation, we define the set of behaviors allowed by Ξ as $Allowed(\Xi) = \bigcap_{\xi \in \Xi} Allowed(\xi)$.

5 Synthesis of Timed Automata from Scenarios

Given a mode graph $\mathcal{M}$ and a set of TES, Ξ, the objective is to build a timed automaton $\mathcal{A}$, such that each behavior that is allowed and covered by Ξ corresponds to a run of $\mathcal{A}$, and vice versa. The automaton should contain the smallest possible number of locations.

Observe that the constructed timed automaton cannot, in general, be replaced by a suitably annotated mode graph. For instance, if the original mode graph contains a cycle to allow multiple attempts to enter a PIN, just adding time annotations would not let us express that the number of attempts must not exceed three, and that all must take place within a given time. That would require adding extensions to the conventional formalism of timed automata.

As another example consider the mode graph and the set of TES $\Xi_1 = \{\xi_1, \xi_5\}$ of Fig. 3. The goal is to construct an automaton, which captures *all* and *only* those behaviors that are both covered and allowed by Ξ_1. Annotating the "relevant" parts of the mode graph with appropriate time constraints, and hence

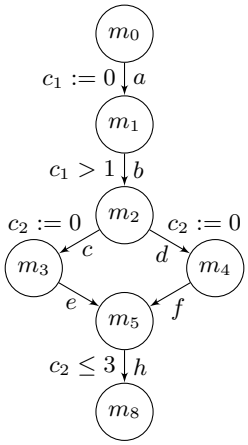

Fig. 4. A timed automaton

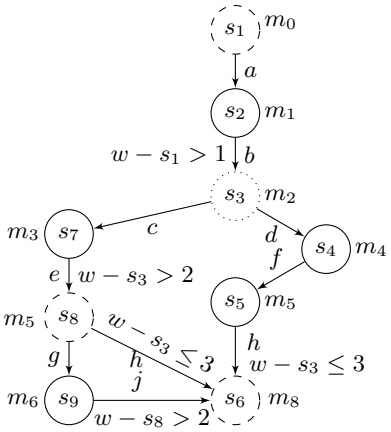

Fig. 5. A TAG synthesized from $\{\xi_1, \xi_3, \xi_4, \xi_5, \xi_6\}$ of Fig. 3

obtaining the automaton of Fig. 4, for instance, would create unsatisfactory results: the automaton would also "show" the behaviors in $\{(abceh, w_1w_2w_3w_4w_5) \mid \forall_{1<i\leq 5}\ w_i > w_{i-1}\}$ which are clearly not covered by Ξ_1.

A requirement of the synthesis algorithm is that the overall automaton have only one initial location and one final location. Moreover, the automaton should be *connected*: each location must be reachable from the initial location and there must be a path from each location to the final location.

We introduce five criteria that a set of TES must satisfy in order to make generation of such an automaton feasible. A set of TES that complies with these criteria is called *complete*.

Given a mode graph $\mathcal{M} = (M, m_0, m_f, \Sigma, T)$, a set Ξ of TES is *complete* if:

1. Every $\xi \in \Xi$ is compatible with $\mathcal{M}$.
2. For every $\xi \in \Xi$, either $initial\_mode(\xi) = m_0$ or there exist $\xi_1, \xi_2, ..., \xi_j \in \Xi$, such that $\xi_j = \xi$ and:
   - $initial\_mode(\xi_1) = m_0$,
   - $initial\_mode(\xi_k) = final\_mode(\xi_{k-1})$, for each $\xi_k,\ 1 < k \leq j$.
3. For every $\xi \in \Xi$, either $final\_mode(\xi) = m_f$ or there exist $\xi_1, \xi_2, ..., \xi_j \in \Xi$, such that $\xi = \xi_1$ and:
   - $final\_mode(\xi_j) = m_f$,
   - $final\_mode(\xi_{k-1}) = initial\_mode(\xi_k)$, for each $\xi_k,\ 1 < k \leq j$.
4. For every $\xi \in \Xi$, $initial\_mode(\xi) \neq final\_mode(\xi)$. Moreover, there is no sequence of TES $\xi_1, \ldots, \xi_j$ such that $\xi = \xi_1$, $final\_mode(\xi_{k-1}) = initial\_mode(\xi_k)$ for $1 < k \leq j$, and $final\_mode(\xi_j) = initial\_mode(\xi)$.
5. All occurrences of an event e must have (textually) identical time annotations.

Considering the mode graph of Fig. 3, the set $\Xi_1 = \{\xi_1, \xi_5\}$ is not complete. But it can be made complete, either by removing ξ_5 or by adding ξ_6. The automaton of Fig. 4 corresponds to the complete set of TES $\Xi_2 = \{\xi_1, \xi_5, \xi_6\}$.

Given a set of TES, checking that the set is complete (based on the criteria defined above) is straightforward. In the rest of the paper, we assume that the given set of TES is complete.

Next, we formally define the problem.

Given a mode graph $\mathcal{M} = (M, m_0, m_f, \Sigma, T)$ and a complete set of TES $\Xi = \{\xi_1, \xi_2, ..., \xi_n\}$, the goal is to construct an acyclic, minimal automaton $\mathcal{A}$ such that: $\mathcal{A}$ has a run on behavior $\mathcal{B}$ iff $\mathcal{B} \in \mathit{Allowed}(\Xi) \cap \mathit{Covered}(\Xi)$.

6 Constructing Locations and Transitions

Our synthesis method is implemented in two major steps. First, the timed events in each $\xi_i \in \Xi$ along with the mode graph $\mathcal{M}$ are used to build a graph, which we call a *time-annotated graph*. The graph consists of nodes[5] connected by time-annotated transitions. Second, clocks are allocated and time annotations are rewritten in terms of these clocks: the result is a timed automaton $\mathcal{A}$.

Before describing our method, we formally define *time-annotated graphs*. Let P be a set of labels, e.g., atomic propositions, etc.

A *time-annotated graph* (*TAG*) is a tuple $G = \langle E, Q, q^0, q^f, R, L \rangle$, where

- E is a finite alphabet;
- Q is the (*finite*) set of nodes;
- $q^0 \in Q$ is the initial node;
- $q^f \in Q$ is the final node;
- $R \subseteq Q \times Q \times E \times 2^{\Phi(Q)}$ is the set of transitions of the form (q, q', a, ϕ), where ϕ is a set of time annotations of the form $w - s \sim a$, where $s \in Q$;
- $L : Q \to P$ is a total function that maps each node to a label.

When we construct a time-annotated graph from a set of scenarios and a mode graph $\mathcal{M} = (M, m_0, m_f, \Sigma, T)$, we will use $E = \Sigma$ and $P = M$.

The first step of our synthesis is preceded by a preprocessing step in which the following three tasks are performed on the set of TES Ξ:

1. For each TES $\xi = \langle m^{initial}, \psi_1 \dots \psi_n, m^{final} \rangle$, if there is a sequence of TES $\xi_1, \dots, \xi_j$ in Ξ, such that $\mathit{events}(\xi) = \mathit{events}(\xi_1) \oplus \dots \oplus \mathit{events}(\xi_j)$ (where $\oplus$ concatenates sequences), then ξ is removed from Ξ.
   Intuitively, if a TES ξ can be broken up into several parts, each of which is a TES in Ξ, then ξ can be safely discarded.
2. For each TES ξ, both the first and the last element of ξ (which name the initial and final modes) is "marked" as a *Join*. In the constructed graph a *Join* will correspond to a node that may have several incoming transitions and several outgoing transitions.
3. For each TES ξ_i, if there is no TES ξ_k, $k \neq i$, such that $\mathit{final\_mode}(\xi_i) = \mathit{final\_mode}(\xi_k)$, the final element of ξ_i (which names the final mode) are

[5] The nodes will be the locations of the synthesized automaton.

marked as *Fork* (i.e., not a general *Join*: the corresponding node cannot have more than one incoming transition).

After the preprocessing step is performed, Algorithm 1 (see p. 12) constructs the nodes and transitions corresponding to the first TES, ξ_1, thus obtaining the initial TAG, G_1. It then repeatedly takes a partial TAG G_k (constructed so far) and integrates a new TES ξ_{k+1} with G_k to obtain the augmented graph G_{k+1}.

As the algorithm constructs the corresponding nodes and transitions of each ξ_i, it also labels the nodes. A newly constructed node q is labelled with mode m_j, i.e., $L(q) = m_j$, if there is a transition on event e from node s to q such that $L(s) = m_i$, and $(m_i, e, m_j) \in T$. Moreover, the algorithm sets the status of each node in the constructed graph. The status of q will be set to *Open* (*Open*$_f$), if m_j is a *Join* (*Fork*) in ξ_i; otherwise it will be set to *Closed*.

Every time the algorithm chooses a new TES ξ, it takes one whose initial mode is already represented by a node s whose status is not *Closed*. It then tries to identify a common prefix between $events(\xi)$ and the sequence of events in a path beginning at s. In doing so, the algorithm avoids merging (i) a *Closed* node q in the graph, where $L(q) = m$, with a mode m that is a *Join* in ξ, and (ii) an *Open* node q in the graph, where $L(q) = m$, with mode m, if m is not a *Join* in ξ. This avoids introduction of new behaviors that are not covered by Ξ. For example, consider the set of TES $\{\xi_1, \xi_5\}$ in Sec. 5. We already mentioned that the automaton of Fig. 4 is not correct for this set. This is because the node corresponding to mode m_2 in the automaton is *Closed*, but is merged with a *Join* in ξ_5.

Criterion 2 on p. 9 ensures that, as long as Ξ is not empty, it will always contain at least one TES such that there is a node s in the constructed graph such that $L(s) = initial\_mode(\xi)$ and the status of s is not *Closed*. So the algorithm will terminate.

Procedure *rename* (see p. 13) deserves a comment. When the current node is q and a timed event (e, ϕ) of some ξ is processed, we rename t_j in $w - t_j \sim a \in \phi$ to s, where s is the latest predecessor of q in G, such that $L(s) = m_j$. Since ξ is compatible with the mode graph, that predecessor must be one of the states that were visited during the integration of ξ, hence there is no possibility of two paths joining between s and q, hence the "latest predecessor" is well-defined.

Fig. 5 shows the time-annotated graph constructed by Algorithm 1 from the mode graph and the set $\{\xi_1, \xi_3, \xi_4, \xi_5, \xi_6\}$ of TES in Fig. 3. Circles in dashed and dotted lines indicate *Open* and *Open*$_f$ nodes, respectively. Note that the path ab corresponds to the common prefix of ξ_1, ξ_3 and ξ_6, as mode m_2 in ξ_6 is not a *Join*, but a *Fork* (task 3 of the preprocessing step on p. 10).

The graph, G, built by Algorithm 1 has the following properties:

1. It is acyclic: we never introduce a transition from a node to its predecessor.
2. By construction, every scenario corresponds to a partial path in the graph, and every path can be partitioned into partial paths that correspond to scenarios.
3. By construction, several nodes in G might be labelled m, but only one of them will not be *Closed*.

Algorithm 1: Building nodes and transitions with time annotations

Input : A mode graph $\mathcal{M} = (M, m_0, m_f, \Sigma, T)$, and a complete, non-empty set of TES $\Xi = \{\xi_1, ..., \xi_n\}$
Output: Time-annotated graph $G_n = \langle E_n, Q_n, q^0, q^f, R_n, L_n \rangle$

$k := 0$; $E_0 := \emptyset$; $Q_0 := \emptyset$; $R_0 := \emptyset$; $L_0 := \emptyset$;
create a new node s: $Q_0 := \{s\}$;
set the status of s to *Open*;
add label m_0 to node s: $L_0 := \{(s, m_0)\}$;
while $\Xi \neq \emptyset$ **do**
  $E_{k+1} := E_k$; $Q_{k+1} := Q_k$; $R_{k+1} := R_k$; $L_{k+1} := L_k$;
  choose a $\xi = \langle m^{initial}, \psi \dots \psi_l, m^{final} \rangle$ in Ξ, such that there is a node s in Q_{k+1} whose status is not *Closed* and $L_{k+1}(s) = m^{initial}$;
  $\Xi := \Xi \setminus \{\xi\}$;
  $s_c := s$; // s_c always indicates the current source
  $j := 1$;
  // Identify a common prefix:
  while $j \leq l \wedge \psi_j = (e, \phi) \wedge (m, e, m') \in T \wedge m'$ *is not a Join in* $\xi \wedge (s_c, q, e, rename(s_c, \psi_j, G_{k+1})) \in R_{k+1} \wedge q$ *is not Open* **do**
    $s_c := q$;
    if $j = l$ **then**
      set the status of q to $Open_f$; // status of m^{final} in ξ
    $j := j + 1$;
  // Create the rest of the nodes and transitions (if any):
  while $j \leq l$ *and* $\psi_j = (e, \phi)$, *where* $(m, e, m') \in T$ **do**
    $\phi' := rename(s_c, \psi_j, G_{k+1})$;
    if $j = l$ *and there is a node* $q \in Q_{k+1}$ *such that* $L(q) = m'$ *and* q *is Open* **then**
      create a new transition $r := (s_c, q, e, \phi')$; // last transition for ξ_i
    else
      create a new node q: $Q_{k+1} := Q_{k+1} \cup \{q\}$;
      add label m' to node q: $L_{k+1} := \{(q, m')\} \cup L_{k+1}$;
      create a new transition: $r := (s_c, q, e, \phi')$;
      if $j = l$ **then**
        if *the status of* m^{final} *in* ξ *is Fork* **then**
          set the status of q to $Open_f$;
        else
          set the status of q to *Open*;
      else
        set the status of q to *Closed*;
        $s_c := q$;
    $R_{k+1} := \{r\} \cup R_{k+1}$; $E_{k+1} := E_{k+1} \cup \{e\}$;
    $j := j + 1$;
  $k := k + 1$;
$q^f := f$, where f is the node with no outgoing transitions; // property (6)

Procedure rename(location q, timed event (e, ϕ), TAG G)

$\phi' := true$;
foreach $w - t_i \sim a \in \phi$ **do**
  $\phi' := \phi' \wedge w - s \sim a$, where s is the nearest predecessor of q in G such that $L(s) = m_i$;
Return ϕ';

4. For every ξ added to the graph the node that corresponds to $initial\_mode(\xi)$ is already in the graph. So the partial path corresponding to ξ will be connected to the graph at node s, i.e., every node will be reachable from the initial one.
5. The order in which the TES are selected from Ξ does not affect the shape of the constructed graph. By property 3 at most one node with a label m is not *Closed*. We connect TES only on nodes that are *Open* or $Open_f$, and the merging of prefixes does not depend on ordering.
6. There must be at least one node with no outgoing transitions, because the graph is finite and acyclic. Each such node is labelled with m_f, because the set of TES is complete (criterion 3 on p. 9 implies that a node whose label is not m_f must have outgoing transitions). But by property (3) at most one node with label m_f is not *Closed*. So all the TES whose final mode is m_f must join at this node. Therefore the final node is unique.
7. Property 6 and criterion 3 on p. 9 imply that the final node can be reached from all the other nodes in the graph.
8. When one considers the labels of the nodes, every run of G is also a run of the mode graph. This is because a transition from s to q is introduced only if there is a transition from $L(s)$ to $L(q)$ in the mode graph.

The TAG constructed by the algorithm might not yet be minimal: there might exist paths ending in the same (*Open*) node that have common suffixes. Such common suffixes should be merged, with appropriate attention given to the status of nodes in order to avoid introduction of new behaviors: the process is very similar to that of merging prefixes.

The resulting automaton (Sec. 7) will be minimal, in the sense that merging any two nodes or transitions would create a cycle or introduce a new behavior.

7 Building the Target Timed Automaton

Algorithm 1 generates the locations and time-annotated transitions of the overall timed automaton in the form of a TAG. The next step of our synthesis method is to transform this TAG to the final automaton. For this, the time annotations of the graph must be replaced by clock operations, i.e., clock resets and clock constraints. There are three tasks: (i) determining the number of clocks required, (ii) identifying transitions at which each clock must get reset and adding the resets, (iii) rewriting the time annotations of the graph as clock constraints. The three tasks mentioned above are performed by Algorithm 2.

Algorithm 2: Generating clock resets and constraints

Input : A time annotated graph $G = \langle E, Q, q^0, q^f, R, L\rangle$
Output: A timed automaton $A = \langle E, Q, q^0, q^f, C, \Theta\rangle$

$C := \emptyset$; $\Theta := \emptyset$; $clock\_assign := \emptyset$;
foreach *transition* $(q_1, q_2, e, \phi) \in R$ **do**
  foreach *time annotation* $w - s \sim a \in \phi$ **do**
    if $(s, c_i) \notin clock\_assign$, *for some clock* c_i **then**
      create a new clock c_i: $C := C \cup \{c_i\}$;
      $clock\_assign := clock\_assign \cup \{(s, c_i)\}$;
foreach *transition* $r = (q_1, q_2, e, \phi) \in R$ **do**
  $\lambda := \emptyset$; $\delta := \emptyset$;
  if $(q_1, c_i) \in clock\_assign$, *for some clock* c_i **then**
    $\lambda := \{c_i\}$;
  foreach *time annotation* $w - s \sim a \in \phi$ **do**
    $\delta := \delta \cup \{(c_j \sim a)\}$, where $(s, c_j) \in clock\_assign$

The dominance assumption and our construction ensure that in the constructed TAG every path leading to a transition r annotated with $w - s \sim a$ must have previously passed through location s. Therefore, the clock assigned to s is *well defined*, i.e., it will always be initialized (reset) before it is used for the first time.

The number of clocks allocated by this algorithm may be greater than strictly necessary. Moreover, some of the resets may be redundant. A method for computing an optimal allocation of clocks, too involved to present here, is described in another paper [13].

After the construction is completed, we identify the final location with the initial location. This allows runs to be infinite. The dominance assumption ensures that the value of a clock from a previous iteration will not be used before it is reset in the current iteration.

From property 2 on p. 11 it follows that the set of runs of the final automaton will correspond exactly to all those behaviours in $B_{\mathcal{M}}$ that are covered by Ξ. Each such behaviour will also be allowed by Ξ. This follows from our assumption that all the occurrences of a particular event in the members of Ξ have exactly the same time annotations: our construction ensures that all the supbaths in the automaton that correspond to those partial behaviours that are relevant to a particular scenario will have equivalent constraints.

Fig. 6 shows the minimal TAG and the automaton synthesized from the two TES in Fig. 2. Without the optimization step mentioned at the end of Sec. 6 the automaton would include two additional states and two more transitions.

8 The Class of Synthesized Timed Automata

Each of our synthesized timed automata has the following properties:

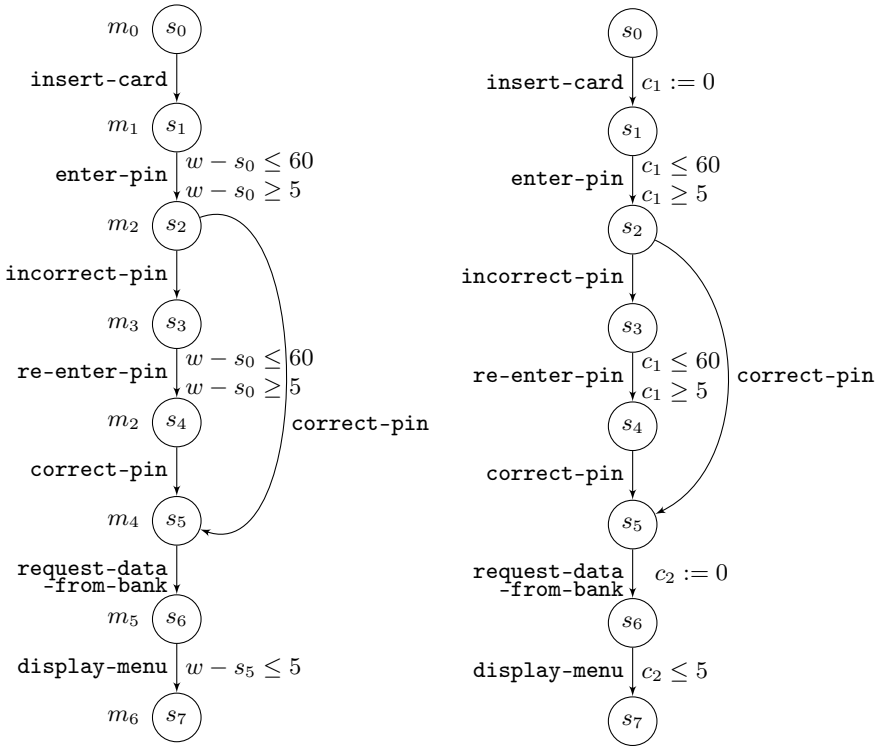

Fig. 6. The synthesized TAG and the automaton for the initial behavior of the ATM

- It has a unique initial location, q^0. There is a path from q^0 to every location.
- Each clock c_j is always reset upon leaving a unique state s_j.
- A clock constraint on a transition r that leaves a state s can refer only to a clock that has been reset upon leaving a state that dominates s.

We call the class of such timed automata TA_{DS}. All automata in TA_{DS} share some interesting properties. These properties allowed us to formulate a clock allocation algorithm whose results are, for all practical purposes, optimal: the details are discussed in another paper [13] (its theoretical underpinings are examined elsewhere [12]). The pivotal role of the dominance assumption makes the class TA_{DS} somewhat restricted, but it can be used to model interesting properties of real-time systems, e.g., safety properties such as *bounded-response* and *bounded-invariance*.

For instance, Fig. 7 shows a variant of a monitor automaton [10], which checks the property that "a is always followed by b within at most 10 time units". Fig. 8 shows an automaton that checks a bounded-invariance property that "b will never occur within 10 time units after a's occurrence".

Periodic behaviors can also be specified quite easily. For instance, the automaton shown at the top of Fig. 9 [1] can be simulated by an automaton in

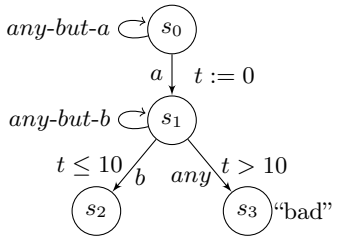

Fig. 7. Bounded response

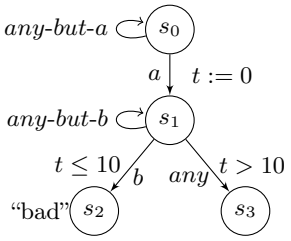

Fig. 8. Bounded invariance

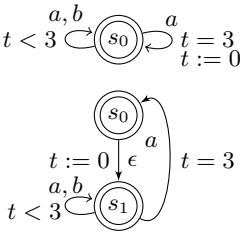

Fig. 9. Periodic behavior

TA_{DS} (shown at the bottom of Fig. 9), by introducing a new location and a silent transition.

9 Related Work

Synthesizing formal models of systems from scenarios has been an active area of research in the last two decades. The research has been focused mostly on formalizing scenarios and developing techniques for scenario-based synthesis of formal models. For scenarios to be used in automatic synthesis methods, they must be both expressive and formal. Choosing the right level of abstraction for representing scenarios is a very important step in this process. Scenarios should formally specify the high level behavior of systems, without going into unnecessary details. The choice of formalisms for representing scenarios also has a great impact on the synthesis method that will be used for building formal models from them.

Recently, Event Sequence Charts (ESCs) have been proposed for expressing scenarios of systems, and a method for synthesizing finite state machines from ESCs has been introduced [8]. Our TES are different from ESCs: ESCs use the notion of monitored, term and controlled variables along with modes, all borrowed from SCR (Software Cost Reduction) [8]. The relation between various actions in ESCs and transitions of mode diagrams are specified by numeric labels. Our Timed Event Sequences accompanied by mode graphs provide a somewhat higher-level representation of scenarios. Our synthesis method is also different, in that it constructs timed automata and not finite state machines.

Various extensions of Message Sequence Charts (MSCs) with time constraints have been proposed for describing scenarios for real-time systems [3]. Different notations for modeling time and expressing timing requirements are used in each of these extensions, e.g., timers with reset and time-outs, and delay intervals for events and activities [2]. UML features sequence diagrams: timing constraints are included by drawing message arrows or by including timing markers [11].

Most of the work on scenario-based synthesis of formal models has been focused on synthesizing state machine models [6, 16, 15]. While the resulting models are useful for reasoning about the overall behavior of systems, they are less useful for reasoning about time and behavioral properties related to time.

To the best of our knowledge the problem of constructing timed automata from a set of scenarios has been addressed only by Somé et al [14]. However, it is not very clear how clocks, clock resets, and clock constraints are generated, and to which transitions the clock resets are assigned. Moreover, the use of contradictory scenarios may cause "unwanted non-determinism" in the constructed automata, which seems undesirable. The authors use the concept of "characteristic conditions" for generating locations and identifying identical locations, but the concept is not formally defined: in particular, it is not obvious whether clock constraints are also considered as characteristic conditions. Moreover, arbitrary variables (e.g., a variable that counts the number of attempts for entering a PIN) are allowed in the constructed automata: this is beyond the conventional formalism of timed automata, which can feature only clock variables [1].

By contrast, in our approach we formally define a set of criteria for a complete set of scenarios from which a synthesis to a timed automaton is possible. We use modes, which are formally defined, as location labels, and use these labels for identifying identical locations. Our algorithm precisely determines the transitions along which each clock must be reset and the transitions where clock constraints must be added.

There has been also some work in scenario-based synthesis of *parametric* timed automata [7], where scenarios with parametric timing constraints in the form of upper and lower time bounds are considered. Allowing parametric constraints in scenarios makes the corresponding synthesis methods difficult to scale up to larger problems, though there are some indications that this might not be a problem in practice [4].

10 Conclusions

We presented a new technique for specifying the required behaviors of real-time systems in terms of a mode graph and a set of scenarios, and of generating a timed automaton from these scenarios.

Intuitively, scenarios put time constraints on the possible partial behaviors of systems. We have used *Timed Event Sequences* to formally describe scenarios.

We have presented a synthesis method for generating a minimal, acyclic timed automaton from a set Ξ of such scenarios, provided that Ξ satisfies our completeness requirements (see Sec. 5). Each behavior allowed by a scenario in Ξ can be viewed as a run of this timed automaton. Moreover, every run of the automaton corresponds to a behavior that is both covered and allowed by the set of scenarios (see Sec. 4).

Our algorithm for constructing a timed automaton comprises two steps. The first step is to generate a time-annotated graph which includes the locations and transitions of the target timed automaton, where transitions are augmented with time annotations from scenarios. Having constructed the skeleton of the target timed automaton, the algorithm performs the second step: clock allocation. The time annotations are used to determine the number of clocks and their reset locations in the target timed automaton, and then to transform the time annotations to clock constraints. The clock allocation algorithm presented here is not

optimal: the description of a more sophisticated version is presented in another paper [13].

11 Acknowledgements

The authors would like to thank one of the anonymous referees of an earlier version of the paper for detailed and helpful comments.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (Apr 1994)
2. Alur, R., Holzmann, G.J., Peled, D.A.: An analyzer for message sequence charts. Software - Concepts and Tools 17(2), 70–77 (1996)
3. Ben-Abdallah, H., Leue, S.: Timing constraints in Message Sequence Chart specifications. In: Togashi, A., Mizuno, T., Shiratori, N., Higashino, T. (eds.) FORTE. IFIP Conference Proceedings, vol. 107, pp. 91–106. Chapman & Hall (1997)
4. Cimatti, A., Palopoli, L., Ramadian, Y.: Symbolic computation of schedulability regions using parametric timed automata. In: RTSS (2008)
5. Clarke, E.M., Klieber, W., Novácek, M., Zuliani, P.: Model checking and the state explosion problem. In: Tools for Practical Software Verification, LASER, International Summer School 2011, Elba Island, Italy, Revised Tutorial Lectures. pp. 1–30 (2011), https://doi.org/10.1007/978-3-642-35746-6_1
6. Damas, C., Lambeau, B., Roucoux, F., van Lamsweerde, A.: Analyzing critical process models through behavior model synthesis. In: Proceedings of the 31st International Conference on Software Engineering. pp. 441–451. IEEE Computer Society (2009)
7. Giese, H.: Towards Scenario–Based Synthesis for Parametric Timed Automata. In: Proceedings of the 2nd International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools (SCESM), Portland, USA (2003)
8. Heitmeyer, C.L., Pickett, M., Leonard, E.I., Archer, M.M., Ray, I., Aha, D.W., Trafton, J.G.: Building high assurance human-centric decision systems. Autom. Softw. Eng. 22(2), 159–197 (2015)
9. Lengauer, T., Tarjan, R.E.: A fast algorithm for finding dominators in a flowgraph. ACM Trans. Program. Lang. Syst. 1(1), 121–141 (Jan 1979)
10. Nicolescu, G., Mosterman, P.J.: Model-Based Design for Embedded Systems. CRC Press, Inc., Boca Raton, FL, USA, 1st edn. (2009)
11. Rational Software: Unified Modeling Language, version 1.1 (September 1997)
12. Saeedloei, N., Kluźniak, F.: Clock allocation in timed automata and graph colouring. http://www2.cs.siu.edu/~neda/report3.pdf
13. Saeedloei, N., Kluźniak, F.: Optimal clock allocation for a class of timed automata. http://www2.cs.siu.edu/~neda/report2.pdf
14. Somé, S., Dssouli, R., Vaucher, J.: From scenarios to timed automata: Building specifications from users requirements. In: Proceedings of the Second Asia Pacific Software Engineering Conference. pp. 48–57. IEEE Computer Society (1995)
15. Uchitel, S., Brunet, G., Chechik, M.: Synthesis of partial behavior models from properties and scenarios. IEEE Trans. Software Eng. 35(3), 384–406 (2009)
16. Uchitel, S., Kramer, J., Magee, J.: Synthesis of behavioral models from scenarios. IEEE Trans. Softw. Eng. 29(2), 99–115 (Feb 2003)