Noname manuscript No.
(will be inserted by the editor)

Scenario-Based Verification of Real-Time Systems Using UPPAAL

Sandie Balaguer · Alexandre David · Kim G. Larsen · Shuhao Li · Brian Nielsen · Saulius Pusinskas

Received: date / Accepted: date

Abstract This article proposes two approaches to tool-supported automatic verification of dense real-time systems against scenario-based requirements, where a system is modeled as a network of timed automata (TAs) or as a set of driving live sequence charts (LSCs), and a requirement is specified as a separate monitored LSC chart.

We make timed extensions to a kernel subset of the LSC language and define a trace-based semantics. By translating a monitored LSC chart to a behavior-equivalent observer TA and then non-intrusively composing this observer with the original TA-modeled real-time system, the problems of scenario-based verification reduce to computation tree logic (CTL) real-time model checking problems. When the real-time system is modeled as a set of driving LSC charts, we translate these driving charts and the monitored chart into a behavior-equivalent network of TAs by using a "one-TA-per-instance line" approach, and then reduce the problems of scenario-based verification also to CTL real-time model checking problems. We show how we exploit the expressivity of the TA formalism and the CTL query language of the real-time model checker UPPAAL to accomplish these tasks. The proposed two approaches are implemented in the UPPAAL tool and built as a tool chain, respectively. We carry out a number of experiments with both verification approaches, and the results indicate that these methods are viable, computationally feasible, and the tools are effective.

Keywords Real-time system · Modeling · Timed automata · Scenario · Live sequence chart · Verification

This work has been supported by the ICT competence center CISS (Center for Embedded Software Systems) and the advanced technology platform DaNES (Danish Network for intelligent Embedded Systems).

Alexandre David, Kim G. Larsen, Shuhao Li (✉), Brian Nielsen, Saulius Pusinskas
CISS, Department of Computer Science, Aalborg University, Aalborg, Denmark
E-mail: {adavid, kgl, li, bnielsen, saulius}@cs.aau.dk

Sandie Balaguer
LSV, ENS Cachan/INRIA, Cachan Cedex, France
E-mail: balaguer@lsv.ens-cachan.fr

1 Introduction

A model checker typically needs two inputs: a model that characterizes the state/transition behaviors of a finite state concurrent system, and a temporal logic formula that specifies the property of interest. For real-time systems, a widely used modeling formalism is timed automata (TA) [2], and the temporal logics could be CTL, LTL, TCTL, etc. Various methods, techniques and tools for model checking real-time systems have been developed over the years, and numerous successful stories of applying them to industrial projects have been reported [6].

Temporal logics such as CTL, LTL, TCTL in themselves are rich enough to formalize a wide range of user requirements such as reachability, safety, liveness and responsiveness. However, these logics are difficult to grasp by non-mathematician/logician users, and in most existing real-time model checkers such as KRONOS [41] and UPPAAL [6], they have only incomplete implementations — on one hand, their atomic propositions are interpreted over the semantic states of timed automata and cannot be event occurrences [41,6]; and on the other hand, there are only limited or even no means for straightforward characterization of quantitative timing constraints [41,6].

The first incompleteness as mentioned above implies that these temporal logics describe only *intra*-process (or "state/transition-based") properties, i.e., whether all states ($\Box$) or at least one state ($\Diamond$) along all paths (A) or at least one path (E) of the individual processes or the product process (i.e., the parallel composed system model) satisfy some particular properties. The second incompleteness implies that general form timing requirements such as $\mathsf{E}\Diamond_{1\leq x\leq 3}$ cannot be easily captured. Altogether, we cannot hope to use these temporal logics to characterize event synchronizations, causal relations, or timed scenarios such as "**if** process B sends message m_1 to process A, and C sends m_2 to D (in any order), **then** B **must** send m_3 to C within 1 to 3 time units" intuitively and conveniently.

Live Sequence Chart (LSC) [18] is a visual formalism for scenario-based requirement specification (in this case, an LSC chart is called a *monitored* chart). Similar to the classical Message Sequence Chart (MSC) [20], LSC also describes *inter*-process properties, i.e., how the system processes interact, collaborate and cooperate via message or rendezvous synchronizations. But beyond that, LSC makes essential extensions to MSC by adding modalities. The *existential* and *cold* (resp. *universal* and *hot*) modalities represent the provisional (resp. mandatory) requirements at global (i.e., whole chart) and local (i.e., message, condition, location and cut) levels, respectively. At the global level, an existential (resp. universal) chart specifies restrictions over at least one satisfying (resp. all possible) system runs. At the local level, for example, a cold condition may be violated and thus lead to a "graceful" chart exit, whereas a hot one must be satisfied and otherwise will indicate an error. The power of LSC lies in that a universal LSC chart can optionally contain a prechart, which specifies the scenario which, if successfully executed (or matched), forces the system to satisfy the scenario given in the actual chart body (i.e., the main chart). Furthermore, the LSC language is unambiguous because it has strictly defined semantics, e.g., the executable (operational) semantics [18] and the trace-based semantics [11,26].

We envisage LSC as a nice complement to the intra-process property specification languages of existing (real-time) model checkers:

- *Intuitiveness*. As a visual formalism, LSC is more intuitive in capturing complex user requirements than the text-form temporal logics;
- *Scenario characterization*. Compared with many temporal logics whose atomic propositions are restricted to be state formulas, LSC has the necessary language constructs (e.g., message and conditional synchronization) to describe process interactions and thus enable the characterization of a variety of causality and non-trivial scenarios. In particular, LSC can be extended to describe timed scenarios;
- *Counterexample display*. In conventional temporal logic model checking, even if a counterexample is due to an inappropriately specified requirement on a correct system model, one has to debug the model to find out the specification error. LSC improves on this by providing the possibility of tracing the counterexamples also back to the visual, scenario-based requirement specifications, and thus facilitates the debugging of both the system models and the user requirements.

In addition to being used as a requirement specification language, LSC can also serve as a scenario-based behavioral modeling language (in this case, each LSC chart is called a *driving* chart). A communicating system can be modeled as a set of driving LSC charts, which we call an *LSC system* (*LS*). Scenario-based modeling using LSCs enjoys the advantage of piecewise incremental construction of system models, i.e., new pieces of scenarios can be added into the models during the development process. However, to check whether an LSC-modeled system satisfies a scenario-based requirement is difficult due to the need to consider both the explicitly specified and implicitly allowed behaviors in each scenario, and the interplays among the different scenarios. The problem becomes even more complicated for real-time systems, as time-enriched LSCs may contain subtle timing errors that are difficult to diagnose. Clearly, this verification problem needs powerful analysis methods, techniques and automated tool support.

In an LSC system, the instance lines in the charts can be viewed as parallel composed processes that interact with one another via message or rendezvous synchronizations. This in spirit resembles some formalisms and tools for concurrent system modeling and analysis. Specifically, the real-time model checker UPPAAL [6] operates on a network of interacting timed automata that communicate via handshake and broadcast synchronizations and shared-variable communications. With its features of committed locations, broadcast channels, and boolean and integer variables, UPPAAL is capable of properly mimicking the behaviors of a time-enriched LSC system using timed automata. This opens up the possibility of exploiting the power of UPPAAL for simulating and analyzing scenario-based real-time system models.

A monitored LSC chart captures the user requirement that once the prechart (if any) is successfully matched, then the main chart **must** be matched afterwards. This is a kind of liveness or responsiveness requirement. Coincidentally, the UPPAAL CTL query language has the "leads-to" property pattern $\phi \leadsto \varphi$ which is a shorthand for $\mathsf{A}\Box(\phi \Rightarrow \mathsf{A}\Diamond\varphi)$, stating that whenever ϕ is satisfied, then eventually φ will be satisfied. By automatically transforming a monitored chart into a behavior-equivalent observer timed automaton, we can specify a corresponding $\phi \leadsto \varphi$ property in UP-

PAAL to capture the LSC requirement. By non-intrusively composing the observer timed automaton with a TA-modeled real-time system, we can achieve the effect of using a monitored chart to "spy on" the system behaviors. All these pave way to verifying real-time systems against scenario-based LSC requirements.

In this article we model a real-time system as a state/transition-based system, more precisely a network of timed automata, or as an object interaction-based system, more precisely a set of driving LSC charts. We capture a scenario-based requirement that is to be verified using a separate monitored LSC chart. We aim at tool-supported automatic verification of the system against the requirement (Fig. 1). As mentioned in the previous paragraphs, we notice that the problems of verifying a state/transition-based real-time system (Fig. 1, left part) and an object interaction-based real-time system (Fig. 1, right part) against a scenario-based user requirement can both be reduced to CTL real-time model checking problems. Since UPPAAL has sophisticated data structures and efficient verification algorithms for handling timing constraints, in this article we will employ UPPAAL as our underlying verification engine.

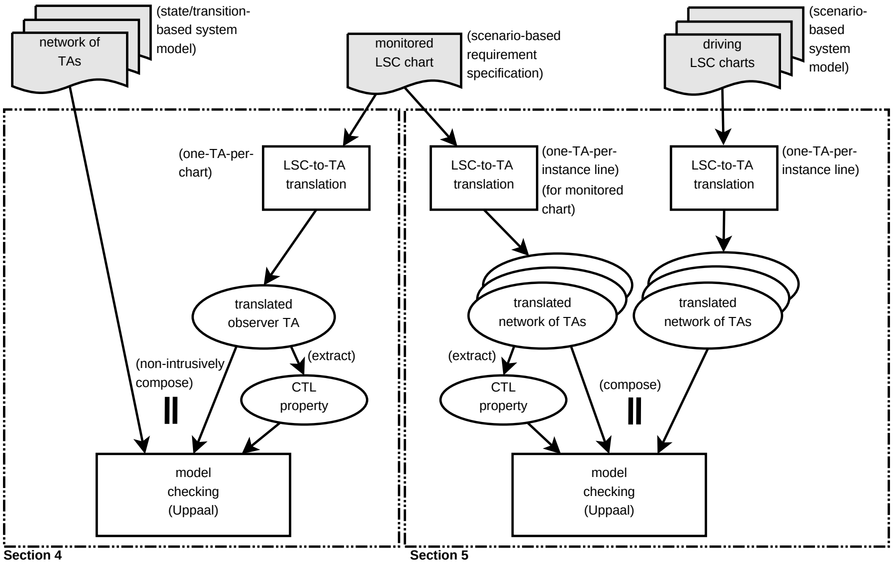


Fig. 1 The overall framework of scenario-based verification of real-time systems

This article is an extended version of previous work at FM'09 (Fig. 1, left part) [28]. In this article, we extend our previous work by complementing (i.e., "horizontally scaling up") the verification framework with systems being modeled as a set of driving time-enriched LSC charts (Fig. 1, right part, partly taken from [29]). We provide lemmas and proofs for the theorems in this article. We add new translation and verification methods for LSC charts under the iterative activation mode. We provide explanations on the prototype tool implementations, and report in more detail some experimental verification results.

1.1 Contributions

The main contributions of this article include:

- We define a kernel subset of the LSC language, make timed extensions to this subset such that it is suitable both for scenario-based behavioral modeling and for scenario-based requirement specification of real-time systems, and we define a trace-based semantics;
- We propose a behavior-equivalent translation of a monitored LSC chart to an observer timed automaton, and propose a method of non-intrusively "observing" an existing TA-modeled real-time system using this observer automaton, thus encoding the problem of verifying state/transition-based real-time systems against scenario-based requirements as a CTL real-time model checking problem. We show how these are integrated into the UPPAAL model checker;
- We propose a behavior-equivalent translation of a driving or monitored LSC chart to a network of timed automata, one for each instance line, and reduce the problem of verifying object interaction-based real-time systems against scenario-based requirements to CTL real-time model checking problems. We implement the LSC-to-TA translator which, together with our LSC editor and the UPPAAL model checker, constitutes a tool chain for scenario-based automatic verification; and
- We conduct experimental evaluations of the proposed approaches, and report the results.

1.2 Organization

Section 2 shows how to model and specify real-time systems using timed automata and temporal logics, respectively, and why scenario-based approaches may come in handy. In Section 3 we define the notations, syntax and semantics of our time-enriched LSC chart. Sections 4 and 5 show how to verify a state/transition-based real-time system and an object interaction-based real-time system against a scenario-based requirement, respectively. Section 6 reports the tool implementations and experimental evaluations of the proposed approaches. Section 7 shows how to horizontally scale up the LSC-to-TA translation methods from invariant mode charts to iterative mode charts. Section 8 discusses some related work on scenario-based verification of real-time systems. Finally, Section 9 concludes this article.

2 Modeling and specification of real-time systems

2.1 Timed automata and computation tree logic

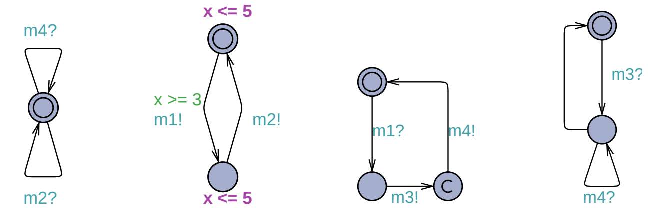
Timed automata (TA) is a popular visual formalism for modeling the state/transition-based behaviors of dense real-time systems. According to Alur and Henzinger [3], the underlying philosophy of TA is that a real-time system can be viewed as a discrete system with clock variables:

- The discrete system is represented as a finite directed graph, where each vertex represents a (control) *location*, and each edge represents an instantaneous *switch* (or discrete jump); and
- The system has a finite set of *clocks* which increase at the same speed and can be reset. Each clock variable keeps track of the elapsed time since last time this clock was reset. Clock variables can be used in boolean expressions to guard the instantaneous switches. Each location may be associated with a clock constraint called *invariant*, specifying the condition under which time can still elapse in this location. [1]

To describe a system which consists of a number of concurrently running processes, a network of timed automata can be constructed, one for each process. These automata are composed in parallel using the operator || . Different automata in the system can synchronize on their common actions [2], and the product automaton has an interleaved execution semantics.

Timed automata in its original form [2] are a simple, concise and yet expressive language. To better support the modeling and automatic verification of real-time systems, various syntactic sugar and extensions are added to the TA formalism. Specifically, UPPAAL [6] strengthens TA with a number of features such as boolean and bounded integer variables, variable constraints and updates, urgent and committed locations, handshake and broadcast channel synchronizations, shared variable communications, etc. Here an *urgent* location is a location where time is frozen (i.e., once an urgent location is entered, it should be exited with zero time delay); and a *committed* location is a special urgent location where the outgoing transitions have higher priority to be taken than those from non-committed ones (Fig. 2(c), the "C"-marked location).

Fig. 2(a)-2(d) give an example of a network of TAs in UPPAAL.



(a) TA *A* (b) TA *B* (c) TA *C* (d) TA *D*

Fig. 2 A real-time system model (network of TAs)

Requirements on TA-modeled real-time systems can be specified using temporal logics such as CTL, LTL or timed variants thereof. For example, UPPAAL uses a fragment of the CTL logic as its property specification language. Atomic propositions

[1] Pragmatically, location invariants are used to allow the system to stay in a location for a limited period of time, and then force it to leave that location.

take the form:

$$ap ::= automaton.location \mid guard\_on\_clocks \mid guard\_on\_variables,$$

and properties can be specified using a number of patterns:

- reachability ($\mathsf{E}\Diamond\phi$);
- safety ($\mathsf{A}\Box\phi$, $\mathsf{E}\Box\phi$); and
- liveness properties ($\mathsf{A}\Diamond\phi$, $\phi \leadsto \varphi$).

In particular the *leads-to* (*responsiveness*) property $\phi \leadsto \varphi$ is a shorthand for $\mathsf{A}\Box(\phi \Rightarrow \mathsf{A}\Diamond\varphi)$, stating that whenever ϕ is satisfied, then eventually φ will be satisfied.

2.2 Scenario-based approaches

Although a lot of properties can be specified by using the above-mentioned property patterns, many others still cannot. Consider a user requirement on the TAs in Fig. 2:

If *we observe that process B sends message m_1 to process C when clock x is no less than 3,* ***then*** *afterwards (and before m_1 can be observed again) we* ***must*** *observe that B sends m_2 to A when x is no less than 2, and C sends m_3 to D (in any order).*

This requirement cannot be specified as a UPPAAL CTL formula or a KRONOS TCTL formula. The reason is that the atomic propositions, which are restricted to be state propositions, do not characterize message passing directly. In other words, they lack the necessary mechanisms for specifying the process interactions and scenarios.

Live Sequence Chart (LSC) is a scenario-based requirement specification language. After extending the LSC language with TA-like clock variables and clock constraints, we notice that the above requirement can be easily captured using LSC (Fig. 3). For instance, the first block of diagrammatic elements $\{m_1, x \geq 3\}$ means that: when message m_1 in the real-time system model is observed, the value of clock x should be no less than 3 at this moment; and if this is the case, then the monitored execution continues, otherwise the prechart (Fig. 3, the outer dashed hexagon) is cold-violated and exited, indicating that this "premise" is not satisfied.

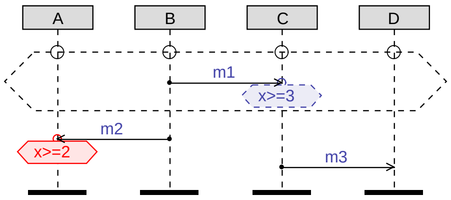


Fig. 3 An LSC chart that expresses a requirement on the real-time system in Fig. 2

Thanks to its *liveness* feature and executable semantics, LSC can also be used to model the scenario-based interaction behaviors of communicating systems. In this case, each LSC chart describes a piece of the "**if** (something happens) **then** (some other thing **must** happen)" style behaviors. A number of driving charts collectively constitute the system model. They characterize how the system processes should interact and collaborate. By allowing LSC to be used both for requirement specification and for system modeling, we may carry out scenario-based validation activities in a much earlier stage of the software development cycle, thus making it possible for developers to focus more efforts on programming/validation-in-the-large rather than programming/validation-in-the-small.

3 Live Sequence Chart: timed extensions and semantics

3.1 Notations and syntax

In this article, LSC in its simplest form is a message-only untimed chart, i.e., there are only language elements of instance lines, locations, messages and precharts/ main charts (Fig. 4).

We make the *synchrony* hypothesis, i.e., system events consume no real time and time may elapse only between events. In this way message synchronizations will be instantaneous, i.e., the sending and receiving of a message are assumed to happen at the same moment in time. Therefore the terms of *message* and (message sending or receiving) *event* will be used interchangeably.

An LSC chart has a role, a type and an activation mode. In this article we consider the roles of:

- *property specification*, i.e., a monitored chart will just "listen to" the messages and read the clock variables in the original system models, but never emit messages to or reset the clocks in those models; and
- *system modeling*, i.e., a driving chart can emit messages and/or reset the clocks when it needs to do so.

A monitored LSC chart could be of the *universal* or *existential* type, whereas a driving chart can only be of the universal type. Since an existential chart is in nature similar to a Message Sequence Chart, in this article we will mainly be interested in universal charts. A driving or monitored universal chart consists of:

- a *main chart* (*Mch*), which specifies what should happen in order for this universal chart to be satisfied (Fig. 4, lower part of the chart [2]); and optionally
- a *prechart* (*Pch*), which specifies the "premise" whose satisfaction triggers the main chart and forces it to be satisfied (Fig. 4, upper part of the chart, i.e., the dashed hexagon area).

If a universal chart has no prechart, then it can be simply treated as having a satisfying prechart. In this article we assume that a universal chart has a prechart. Furthermore, an existential chart consists of only a main chart.

A universal chart has an *activation mode* which determines how often a chart should be activated. In this article we consider:

[2] In the original definition of LSC [18], the main chart of a universal chart should be enclosed within a solid rectangle borderline, whereas that of an existential chart should be within a dashed borderline. For brevity, in this article we omit the borderlines of universal charts. Since no existential chart examples are given in this article, no confusion arises between existential charts and non-prechart universal charts.

- the *invariant* mode, i.e., the prechart is being constantly matched for in the message stream (i.e., for any arriving message, in addition to being monitored by existing prechart copies, it will initiate a new prechart copy), and the main chart will be activated (i.e., a live chart copy will be incarnated and then enforced) whenever the prechart is successfully completed; and
- the *iterative* mode, i.e., as long as the main chart is currently active, the prechart will not be monitored for further satisfaction (until the current "iteration" of the main chart is over).

In the rest of this section and Sections 4 - 6, we consider only the invariant mode LSC charts and their translations. The case of the iterative activation mode will be addressed in Section 7.

3.1.1 Message-only untimed chart

We start with message-only untimed charts. See Fig. 4 for an example.

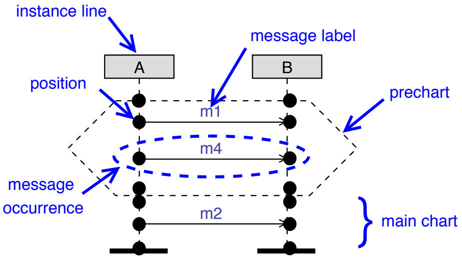


Fig. 4 Anatomy of an example untimed LSC chart L

Each LSC chart describes a particular interaction scenario among a set of processes (or instances, or agents). Given a universal chart L, let $I = inst(L)$ be the set of instance lines in L (Fig. 4, instance lines $\{A, B\}$). Along each instance line $I_i \in I$ there is a finite set of "positions" $pos(L, I_i) = \{0, 1, 2, \ldots, p\_max_{L,I_i}\} \subset \mathbb{N}_{\geq 0}$, which denote the points of communication, computation and synchronization (Fig. 4, black filled circles along A and B). Specifically, along each instance line I_i there are four "standard" positions $StdPos(L, I_i) = \{Pch\_top_{L,I_i}, Pch\_bot_{L,I_i}, Mch\_top_{L,I_i}, Mch\_bot_{L,I_i}\} \subseteq pos(L, I_i)$, denoting the entry/exit points of the prechart/main chart, respectively (Fig. 4, downward ascending positions 0, 3, 4, 6 on instance line A), such that:

- $0 = Pch\_top_{L,I_i} < Pch\_bot_{L,I_i} < Mch\_top_{L,I_i} < Mch\_bot_{L,I_i} = p\_max_{L,I_i}$; and
- $Pch\_bot_{L,I_i} + 1 = Mch\_top_{L,I_i}$. □

The positions of an existential chart L can be defined similarly. Since an existential chart has no prechart, an instance line I_i of it has only two "standard" positions Mch_top_{L,I_i} and Mch_bot_{L,I_i}.

A chart *location* is a position on a certain instance line of the chart. The set of all locations of chart L is denoted as:

$$Loc = loc(L) = \{\langle I_i, p\rangle \mid I_i \in inst(L),\ p \in pos(L, I_i)\}.$$

Since a "standard" position cannot be the end point of a message, the set of all *message-anchoring* locations of L is denoted as:

$$loc_M(L) = \{\langle I_i,\, p\rangle \mid I_i \in inst(L),\, p \in pos(L,I_i) \backslash StdPos(L,I_i)\}.$$

Furthermore, we define a function $psn : loc(L) \to \bigcup_{I_i \in inst(L)} pos(L,I_i)$ to project a location to its **posi**tio**n** on its instance line.

Let L be an LSC chart, and $ML(L)$ be the set of **m**essage **l**abels (or "signals", or "channels" in UPPAAL) of L (Fig. 4, $\{m_1, m_2, m_4\}$). A *message occurrence* $mo = (\langle I_i, p\rangle, m, \langle I_{i'}, p'\rangle) \in loc_M(L) \times ML(L) \times loc_M(L)$ corresponds to instance I_i, while in its position $(p-1)$, sending signal $m \in ML(L)$ to instance $I_{i'}$ at its position $(p'-1)$, and then arriving at positions p and p', respectively (Fig. 4, ellipse-circled portion). We call $lab(mo) = m$ the message label, $head(mo) = \langle I_{i'}, p'\rangle$ and $tail(mo) = \langle I_i, p\rangle$ the message head and tail locations, and $src(mo) = I_i$ and $dest(mo) = I_{i'}$ the source and destination instances, respectively. We use $loc(mo) = \{head(mo), tail(mo)\}$ to denote the message anchoring locations. The set of all message occurrences in chart L is denoted as:

$$\begin{aligned} MO(L) \subseteq \{(\langle I_i, p\rangle, m, \langle I_{i'}, p'\rangle) \in loc_M(L) \times ML(L) \times loc_M(L) \mid \\ p \leq Pch\_bot_{L,I_i} \Leftrightarrow p' \leq Pch\_bot_{L,I_{i'}}\}. \end{aligned}$$

We omit the parameter L in $MO(L)$ (and thus abbreviating it as MO) when it is clear from the context. Furthermore, the projection of $MO(L)$ onto $inst(L) \times ML(L) \times inst(L)$ is denoted as $\Sigma = MA(L)$ ("**m**essage **a**lphabet"), where each letter is a *message* which denotes that a particular signal is sent from one object (instance line) to another. For a given message occurrence, we may overload its "message label" to also denote the corresponding letter in Σ.

This article does not consider concurrent messages (i.e., we assume that any instance line does not send and/or receive two or more messages simultaneously), thus each location can be the end point of at most one message occurrence in the chart.

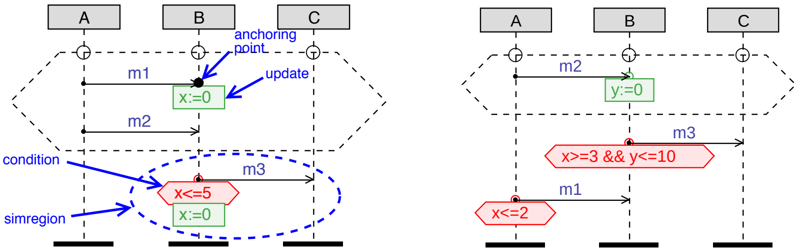
3.1.2 Time-enriched chart

Now we continue to define our timed extensions to the above kernel subset of the LSC language. In our time-enriched LSC charts, there are further elements of (clock) variables, conditions (clock constraints), updates (clock resets) and simregions (i.e., "simultaneous regions"). Fig. 5 gives two example time-enriched LSC charts (for clarity the normal positions on the instance lines are omitted).

Assume that in chart L there is a finite set X of real-valued *clock variables* that range over $\mathbb{R}_{\geq 0}$. A *clock valuation* is a function $v : X \to \mathbb{R}_{\geq 0}$ that maps each clock variable to a non-negative real number, also denoted $v \in {\mathbb{R}_{\geq 0}}^X$.

Let $d \in \mathbb{R}_{\geq 0}$. Notation $(v+d) : X \to \mathbb{R}_{\geq 0}$ means that the clock valuation v is shifted by d such that $\forall x \in X\,.\,(v(x+d) = v(x)+d)$.

A *clock constraint* is of the form $x \bowtie n$ or $x - y \bowtie n$ where $x, y \in X$, $n \in \mathbb{Z}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$. Let $B(X)$ be the set of finite conjunctions over these constraints. A *condition* (or *guard*) is an element from $B(X)$ that spans across (and thus "anchors" on or intersects with) one or more instance lines, denoted $g \in Loc^+ \times B(X)$. Here



(a) Timed chart L_1 (b) Timed chart L_2

Fig. 5 Example time-enriched LSC charts

$Loc^+ = \bigcup_{i=1}^{card(I)} Loc^i$ represents the union of Cartesian products where the number of Loc's ranges from 1 to $card(I)$ (i.e., the cardinality of I). The set of guards in a chart is denoted $G \subset Loc^+ \times B(X)$. The set of anchoring locations of condition g is denoted $loc(g)$. We may omit the location information of a guard when it is not explicitly needed in the context (Fig. 5(b), $\{x \geq 3 \wedge y \leq 10, x \leq 2\}$).

The LSC language constructs such as location, message and condition each have a cold/hot modality (or "temperature"). For example, the temperature of a condition $g \in G$ is denoted $g.temp$. When $temp$ is cold, the condition is enclosed in a dashed hexagon (Fig. 3, $x \geq 3$); when $temp$ is hot, it is enclosed in a solid hexagon (Fig. 5(a), $x \leq 5$). Temperature defines the criticality of a condition, and it also determines the consequence when this condition evaluates to false. While a hot condition must be satisfied, a cold condition may or may not be satisfied. When a hot condition evaluates to false, it indicates a violation of the system requirement. In contrast, a cold condition evaluating to false merely induces a "graceful" exit from the chart. The temperature of a condition may be either hot or cold in a main chart. However, it can only be cold in a prechart. The reason is that a prechart itself does not enforce anything.

In the original definition of LSC [18], many language constructs have their hot/cold modalities. For simplicity, in this article we distinguish between the cold and hot temperatures only for conditions, and assume hot as the default temperature for other constructs.

A *clock reset* is of the form $x := 0$ where $x \in X$. An *assignment* (or update) is the union of a finite set of clock resets (also written as the set of clocks to be reset) that spans across one or more instance lines, denoted $a \in Loc^+ \times 2^X$. The set of all assignments in the chart is denoted $A \subset Loc^+ \times 2^X$. The set of anchoring locations of assignment a is denoted $loc(a)$. The location information of an assignment may also be omitted if it is not explicitly needed in the context (Fig. 5(b), $y := 0$). Furthermore, we can also view $a \in A$ as a transformer on the functions of clock valuations, and as such the new valuation of v after assignment a is denoted by $v' = a(v)$.

Unlike in untimed charts where a message occurs all by itself, in time-enriched LSCs, each message occurrence mo can be optionally associated with a condition g and/or an assignment a. Notationally the condition and/or assignment is anchored to an end point of the message (Fig. 5(a), the two anchoring points on m_1 and m_3). A message occurrence and the condition and/or assignment attached thereto (if any) constitute an atomic step of LSC execution (Fig. 5(a), the ellipse area), i.e., they take place at the same moment in time. As inspired by [25], we call such a structure of message occurrence/condition/assignment a *simultaneous region* (simregion). The intuitive meaning of message synchronization $[g]\, mo\,/\,a$ from location $\langle I_i, p\rangle$ to $\langle I_{i'}, p'\rangle$ is that, if when *mo* occurs, the clock valuation v satisfies g, then this synchronization can fire; and immediately after the firing, v will be updated according to a.

When a simregion s contains a message, then s is called a *message simregion* (Fig. 5(a), the ellipse area). If s has no message occurrence, then s consists of a condition test, or an assignment, or both of them combined and anchored together. In this case, s is called a *non-message simregion* (Fig. 8(c)). For such a simregion, we adopt the As-Soon-As-Possible (ASAP) semantics for its firing, i.e., the condition test (if any) will be evaluated immediately after the execution of the previous simregion, and the update (if any) follows immediately.

When the condition, message occurrence or assignment part is missing in a simregion, we denote that part as ε. Since such an “absence” does not correspond to any location, we let $loc(\varepsilon) = \emptyset$.

Definition 1 (simregion) A *simregion s* is a tuple of LSC condition, message occurrence, and assignment, $s = (g, mo, a) \in (G \cup \{\varepsilon\}) \times (MO \cup \{\varepsilon\}) \times (A \cup \{\varepsilon\})$, which is subject to the following constraints:

- *common anchoring point.* If $mo \neq \varepsilon$, then $(g \neq \varepsilon \Rightarrow loc(g) \cap loc(mo) \neq \emptyset) \wedge (a \neq \varepsilon \Rightarrow loc(a) \cap loc(mo) \neq \emptyset)$; if $mo = \varepsilon$, then $(g \neq \varepsilon \wedge a \neq \varepsilon \Rightarrow loc(g) \cap loc(a) \neq \emptyset)$;
- *non-emptiness.* $(g \neq \varepsilon) \vee (mo \neq \varepsilon) \vee (a \neq \varepsilon)$; and
- *no overlapping with other simregions.* $\forall s' = (g', mo', a')\,.\,((loc(g) \cup loc(mo) \cup loc(a)) \wedge (loc(g') \cup loc(mo') \cup loc(a')) \neq \emptyset \Rightarrow (g = g') \wedge (mo = mo') \wedge (a = a'))$. □

The set of all simregions in a chart is denoted $SR \in 2^{(G\cup\{\varepsilon\})\times(MO\cup\{\varepsilon\})\times(A\cup\{\varepsilon\})}$.

For example, in Fig. 3 there are three simregions $s_1 = (x \geq 3, m_1, \varepsilon)$, $s_2 = (x \geq 2, m_2, \varepsilon)$, and $s_3 = (\varepsilon, m_3, \varepsilon)$. Note that for brevity here we use the message labels to represent the corresponding message occurrences.

For a simregion $s = (g, mo, a) \in SR$, we use $loc(s) = loc(g) \cup loc(mo) \cup loc(a)$ to denote the set of anchoring locations of simregion s. For example, in Fig. 5(a) the circled simregion $s = (x \leq 5,\, m_3,\, x := 0)$ has two anchoring locations $loc(s) = \{\langle B, 5\rangle, \langle C, 3\rangle\}$.

Now that we have presented the necessary basic and composite LSC constructs, we are in place to give the following generic definition of LSC.

Definition 2 (live sequence chart, LSC) A *live sequence chart* is a tuple $L = \langle I, Loc, ML, X, MO, G, A, PchBot\rangle$, where

- $I = inst(L)$ is the set of instance lines in L;
- $Loc = loc(L)$ is the set of locations in L;
- ML is the set of message labels in L;
- X is the set of clocks in L;
- $SR \in 2^{(G\cup\{\varepsilon\})\times(MO\cup\{\varepsilon\})\times(A\cup\{\varepsilon\})}$ is the set of simregions in L, where:

  - $G \subset Loc^{+} \times B(X)$ is the set of guards;
  - $MO \subset Loc \times ML \times Loc$ is the set of message occurrences;
  - $A \subset Loc^{+} \times 2^{X}$ is the set of updates; and
- $PchBot \in \{nil\} \cup Loc^{+}$ is the bottom location vector of the prechart. In particular when L has no prechart $PchBot$ is nil. □

The BNF grammar for our time-enriched LSC language is given in Appendix A. The Class Diagram of our time-enriched LSC language is presented in Fig. 6, where the solid diamond lines and normal lines represent the "composition" and "association" relationships among the LSC constructs, respectively. Note that in Fig. 6 the `Cut` class and its relationships to other classes will be explained in Section 3.2.1.

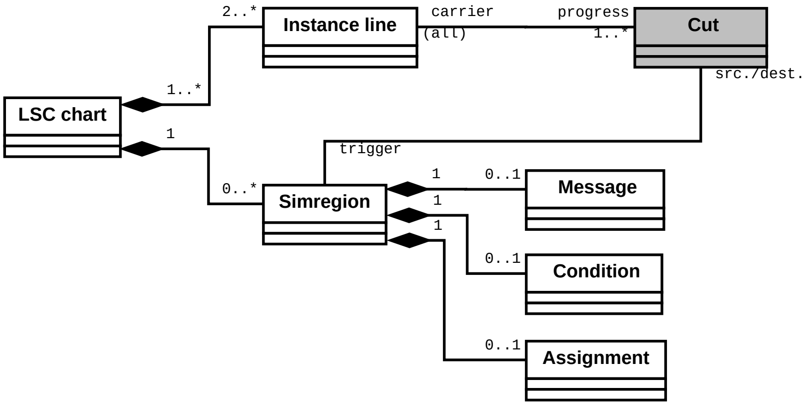


Fig. 6 Class Diagram for time-enriched LSC language

3.2 Trace-based semantics

3.2.1 Semantics for a single universal chart

In an LSC chart L, every location is either associated with a simregion, or it is an entry/exit point of the prechart/main chart. We define a labeling function $\lambda : loc(L) \rightarrow SR \cup \{nil\}$ to map a location of the former type to its corresponding simregion, and a location of the latter type to nil.

Locations in a chart L are preordered ($\leq$) as follows:

- Along each instance line I_i, locations are downward increasing: location l is above $l' \Rightarrow (l \leq l') \wedge \neg(l' \leq l)$; and
- All locations in the same simregion have the same order: $\forall s \in SR, \forall l, l' \in loc(L)$. $(\lambda(l) = s) \wedge (\lambda(l') = s) \Rightarrow (l \leq l') \wedge (l' \leq l)$. □

The preorder relation $\preccurlyeq \subseteq loc(L) \times loc(L)$ is defined as a transitive closure of $\leq$. For example in Fig. 5(b), $\langle A, 0\rangle \preccurlyeq \langle B, 1\rangle \preccurlyeq \langle B, 5\rangle \preccurlyeq \langle C, 4\rangle$ is a preorder of the locations.

A cut represents all the locations along all instance lines that have been progressed so far. It is formally defined as follows.

Definition 3 (cut of an LSC chart) A *cut* of a chart L is a set $c \subseteq loc(L)$ of locations that span across all the instance lines in L and satisfy the properties of:

- *downward-closure*. If a location l is included in cut c, so are all of its predecessor locations: $\forall l, l' \in loc(L) \,.\, ((l \in c \wedge l' \preccurlyeq l) \Rightarrow l' \in c)$; and
- *intra-chart coordination integrity*. If a *Mch_top* position of a certain instance line is included in the cut, then the *Mch_top* positions of all other instance lines are also included in the cut: $\exists l \in loc(L), I_i \in inst(L) \,.\, ((Mch\_top_{L,I_i} \leq psn(l)) \wedge (psn(l) \leq Mch\_top_{L,I_i}) \wedge (l \in c) \Rightarrow \forall l' \in loc(L), I_{i'} \in inst(L) \,.\, ((Mch\_top_{L,I_{i'}} \leq psn(l')) \wedge (psn(l') \leq Mch\_top_{L,I_{i'}}) \Rightarrow l' \in c))$. □

For a cut c, we use $loc(c)$ to denote its *frontier*, i.e., the set of locations that constitute the downward borderline progressed so far. The location where c "cuts" instance line $I_k \in I$ is denoted $loc(c)_{\langle I_k \rangle}$. For example in Fig. 5(a), if c is the cut when the main chart is just entered, then $loc(c) = \{\langle A, 4\rangle, \langle B, 4\rangle, \langle C, 2\rangle\}$, and $loc(c)_{\langle A \rangle} = \langle A, 4\rangle$. The set of all cuts is denoted as *Cuts*.

Given a cut $c \in Cuts$ and a simregion $s \in SR$, we say s is *enabled* at cut c (with respect to the location preorder relation), denoted $c \xrightarrow{s}$, if each anchoring location of s immediately follows a certain location in c. Formally, $\forall l \in loc(s) \,.\, \exists l' \in c \,.\, ((l' \preccurlyeq l) \wedge \neg(l \preccurlyeq l')) \wedge (\nexists l'' \in loc(L) \backslash (c \cup loc(s)) \,.\, (l' \preccurlyeq l'' \wedge l'' \preccurlyeq l))$. For example, in Fig. 5(a) the circled simregion is enabled at the cut when the main chart is just entered. The enabledness of message occurrences can be defined similarly.

A cut c' is an *s-successor* of cut c, denoted $c \xrightarrow{s} c'$, if s is enabled at c (w.r.t. the location preorder), and c' is achieved by adding the set of locations that s anchors at into c. Formally, $c \xrightarrow{s} c' \Leftrightarrow (c \xrightarrow{s}) \wedge (c' = c \cup loc(s))$.

Since a simregion triggers a new cut, this new cut and the original cut can be viewed as the destination and source cuts of the LSC advancement step, respectively. This relationship between the `Cut` and `Simregion` classes is depicted in Fig. 6. We also stress that in Fig. 6 the `Cut` class is singled out in shaded rectangle, because it is a semantic rather than a syntactical concept.

A cut c is *minimal*, denoted $\top$, if it "cuts" each instance line at its top location; and c is *maximal*, denoted $\bot$, if it "cuts" each instance line at its bottom location. The minimal and maximal cuts of the prechart and main chart are denoted $Pch.\top$, $Pch.\bot$, $Mch.\top$ and $Mch.\bot$, respectively. The frontiers of minimal and maximal cuts do not contain simregion anchoring points. Rather the cuts $Pch.\bot$ and $Mch.\bot$ each represent a requirement for compulsory synchronization of all the instance lines in the chart. Thus the preorder relation $\preccurlyeq$ on $loc(L)$ is extended as follows (and finally also extended to its transitive closure):

- All locations in the frontier of the same minimal or maximal cut have the same order: $\forall c \in \{Pch.\top, Pch.\bot, Mch.\top, Mch.\bot\}, \forall l, l' \in loc(c) \,.\, (l \preccurlyeq l') \wedge (l' \preccurlyeq l)$. □

For example, in Fig. 3 the possible cuts are: $\{\}$, $\{s_1\}$, $\{s_1, s_2\}$, $\{s_1, s_3\}$, $\{s_1, s_2, s_3\}$, where e.g. $\{s_1\}$ is a shorthand for the cut where simregion s_1 has just been stepped over. Clearly, cut $\{s_1, s_2, s_3\}$ is the s_3-successor of cut $\{s_1, s_2\}$.

Based on the preorder relation $\preccurlyeq$ on $loc(L)$, we can induce an event (message) partial order relation of the chart.

In order to define a trace-based semantics for LSC, we need to determine what is a "semantic state" of a time-enriched LSC chart. In this article we call such a semantic state an LSC configuration.

Definition 4 (configuration) A *configuration* of an LSC chart is a tuple (c,v), where c is a cut and v is a clock valuation. □

A configuration at the minimal cut $\top$ with all clocks assigned their initial values (e.g., 0's) is called the *initial* configuration.

In each configuration, we can check whether a next coming message violates the event partial order of the chart, and whether a next condition evaluates to true. If in the main chart the event partial order is violated or a hot condition evaluates to false, then it is a *hot violation*. In comparison, if the event partial order is violated in the prechart or a cold condition evaluates to false, then it is a *cold violation*. A hot violation means that some mandatory requirements are not satisfied and therefore there is an error in the system (e.g., an exception rather than expected message occurs in the main chart), whereas a cold violation means that some provisional requirements are not satisfied and therefore the chart can be gracefully exited (e.g., a condition in the prechart evaluates to false).

A universal chart starts from the initial configuration, advances from one configuration to a next one, until a hot violation occurs, or until the chart arrives at a maximal cut configuration and then starts all over again (i.e., to begin a next round execution).

There could be three kinds of advancement steps between two configurations (c,v) and (c',v') of a time-enriched LSC chart:

- *Message synchronization step*. Given a simregion s which consists of an m-labeled message occurrence mo $(m \in \Sigma)$, and optionally a condition g and/or assignment a, there is a message synchronization step $(c,v) \xrightarrow{m} (c',v')$ if:
  - (normal advancement). $c \xrightarrow{s} c'$, $v \models g$, and $v' = a(v)$; or
  - (cold violation). $c' = Pch.\top$, $v' = v$, and either
    - mo is not enabled at cut c in the prechart (w.r.t. the preorder relation); or
    - $v \not\models g \wedge g.temp = cold$;
- *Silent step*. Given a simregion s which consists of a condition g and/or assignment a, there is a silent step $(c,v) \xrightarrow{\tau} (c',v')$ if either
  - (*silent advancement*). $proj_{MO \cup \{\varepsilon\}}(s) = \varepsilon$, $c \xrightarrow{s} c'$, $v \models g$, and $v' = a(v)$; or
  - (*prechart-main chart transition*). $(c = Pch.\bot,\ c' = Mch.\top,\ v' = v)$; or
  - (*main chart-prechart transition*). $(c = Mch.\bot,\ c' = Pch.\top,\ v' = v)$; or
  - c' is reached because an instance line moves to its bottom location in Pch or Mch autonomously (this happens when the instance line will not interact with other instance lines before it reaches its bottom location in Mch or Pch). Formally, there exists an instance I_k such that $v' = v$ and either
    - $loc(c')_{\langle I_k \rangle} = (loc(Pch.\bot))_{\langle I_k \rangle}$, $psn(loc(c')_{\langle I_k \rangle}) = psn(loc(c)_{\langle I_k \rangle}) + 1$, and $loc(c')_{\langle I_i \rangle} = loc(c)_{\langle I_i \rangle}$ for all $I_i \neq I_k$; or

- $loc(c')_{\langle I_k \rangle} = (loc(Mch.\bot))_{\langle I_k \rangle}$, $psn(loc(c')_{\langle I_k \rangle}) = psn(loc(c)_{\langle I_k \rangle}) + 1$, and $loc(c')_{\langle I_i \rangle} = loc(c)_{\langle I_i \rangle}$ for all $I_i \neq I_k$;

– *Time delay step*. There is a time delay step $(c,v) \xrightarrow{d} (c',v')$ where $d \in \mathbb{R}_{\geq 0}$ if: $c' = c$, $v' = v + d$, and whenever there are message occurrences that are enabled at cut c (w.r.t. both the preorder relation and the guard), then after delay d there exists at least one of them that is still enabled at the same cut, i.e., $\exists s \in SR\,.\,(proj_{MO \cup \{\varepsilon\}}(s) \neq \varepsilon) \wedge (proj_{G \cup \{\varepsilon\}}(s) = g) \wedge (\forall d' \in [0,d]\,.\,(c \xrightarrow{s}) \wedge (v + d') \models g)$. □

Similar to the above-mentioned cold violation case, if in the main chart, an m-labeled message violates $\preccurlyeq$, or $(v \not\models g \wedge g.temp = hot)$, then the configuration (c,v) is said to be *hot-violated*, denoted $(c,v) \overset{m}{\not\rightarrow}$.

Definition 5 (run of an LSC chart) A *run* of a time-enriched universal LSC chart is a sequence of configurations $(c^0,v^0) \cdot (c^1,v^1) \cdot \ldots$ that are connected by the advancement steps $(c^i,v^i) \xrightarrow{u_i} (c^{i+1},v^{i+1})$, where $u_i \in (\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})$, $i \geq 0$. □

A transition in Def. 5 carries only a single letter $u \in (\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})$. We extend $\rightarrow$ to $\rightarrow^*$ such that a transition carries a (finite or infinite) word $w \in (\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^* \cup (\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^\omega$.

Let Π correspond to the set of all possible messages that occur in a state/transition-based system model (i.e., a network of timed automata), or be the set of all messages in an object interaction-based system model (i.e., a set of driving universal LSC charts). In the latter case, the message alphabet for the LSC system model $LS = \{L_i \mid 1 \leq i \leq n\}$ is $\Pi = \bigcup_{i=1}^{n} \Sigma_i = \bigcup_{i=1}^{n} MA(L_i)$.

Definition 6 (satisfaction of a prechart/main chart) A timed trace $\gamma \in (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^* \cup (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^\omega$ *satisfies* an LSC prechart or main chart C, denoted $\gamma \models C$, if its restriction $\gamma|_{(\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})}$ has a prefix μ which is the accepted word of a run that starts from the initial configuration and arrives at a maximal cut configuration of C, and no prefix of it ever leads to a hot violation. Formally, $\gamma \models C \Leftrightarrow (\exists \mu \in (\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^*, \xi \in (\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^* \cup (\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^\omega, v' \in {\mathbb{R}_{\geq 0}}^X\,.\,(\gamma|_{(\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})} = \mu \cdot \xi) \wedge (\top, v^0) \overset{\mu}{\rightarrow^*} (\bot, v')) \wedge (\nexists \mu' \in (\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^*, \xi \in (\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^* \cup (\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^\omega, m \in \Sigma, \bullet \in Cuts \times {\mathbb{R}_{\geq 0}}^X\,.\,((\gamma|_{(\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})} = \mu' \cdot m \cdot \xi) \wedge (\top, v^0) \overset{\mu'}{\rightarrow^*} \bullet \overset{m}{\not\rightarrow}))$. □

Definition 6a *A finite trace $\gamma \in (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^*$ satisfies chart C* exactly, *denoted $\gamma \Vdash C$, iff γ satisfies C, and its restriction on $(\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})$ is the accepted word of a run that contains and ends with exactly one maximal cut configuration of C. Formally,* $\gamma \Vdash C \Leftrightarrow (\gamma \models C) \wedge \exists \mu \in (\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^*,\ v' \in {\mathbb{R}_{\geq 0}}^X\,.\,(\gamma|_{(\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})} = \mu) \wedge ((\top, v^0) \overset{\mu}{\rightarrow^*} (\bot, v')) \wedge \nexists \varepsilon \neq \mu' \in (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^*, \varepsilon \neq \xi \in (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^*,\ v'' \in {\mathbb{R}_{\geq 0}}^X\,.\,(\gamma|_{(\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})} = \mu' \cdot \xi) \wedge ((\top, v^0) \overset{\mu'}{\rightarrow^*} (\bot, v''))$. □

Now we define the satisfaction relation for a full universal chart (under the invariant activation mode):

Definition 7 (satisfaction of a universal LSC chart) A timed trace $\gamma \in (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{\omega}$ *satisfies* (passes) a universal chart L, denoted $\gamma \models L$, iff whenever a finite subtrace matches the prechart, then the main chart is matched immediately afterwards. Formally, $\gamma \models L \Leftrightarrow \forall \alpha, \mu \in (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{*}, \beta \in (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{\omega} \,.\, (\alpha \cdot \mu \cdot \beta = \gamma) \wedge (\mu \Vdash Pch) \Rightarrow (\beta \models Mch)$. □

Definition 7a *A timed language Lang* $\subseteq (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{\omega}$ *satisfies chart L, denoted Lang* $\models L$*, iff every word of Lang satisfies the chart. Formally, Lang* $\models L \Leftrightarrow \forall \gamma \in$ *Lang* $. \gamma \models L$. □

As seen above, *Lang* characterizes the system behaviors that respect L.

When L is used as a monitored chart, then for a network S of timed automata, we use $S \models L$ to denote that the timed traces (language) of S satisfy LSC L.

3.2.2 Semantics for a set of driving universal charts

For an LSC system *LS* which consists of a set of driving universal charts $L_1, L_2, \ldots, L_n$, we denote a *cut vector* of *LS* by $\bar{c} = (c_1, c_2, \ldots, c_n)$, and a *valuation* of all of the clock variables in *LS* by v. Each member cut of $\bar{c}$ is denoted by $c_i = (\bar{c})_i$, $1 \leq i \leq n$. We call $(\bar{c}, v)$ a *global configuration* of *LS*.

Let $(\bar{c}, v)$ be a global configuration of an LSC system *LS*. Assume that there are message occurrences $mo_1, \ldots, mo_k$ ($1 \leq k \leq n$, each in a different chart) that are simultaneously enabled at $((\bar{c})_1, v), \ldots, ((\bar{c})_k, v)$, and that these message occurrences are the same message (they have exactly the same message label and the same source and destination instances), i.e., $\exists m \in \Pi, L_j \in LS, I_a, I_b \in inst(L_j) \,.\, \forall 1 \leq i \leq k \,.\, (lab(mo_i) = m) \wedge (src(mo_i) = I_a) \wedge (dest(mo_i) = I_b)$. In this case, these identically labeled message occurrences are said to be *enabled* at global configuration $(\bar{c}, v)$ w.r.t. their respective preorder relations.

Given a global configuration $(\bar{c}, v)$ of *LS* and a message $m \in \Pi$, there is a message synchronization step $(\bar{c}, v) \xrightarrow{m} (\bar{c}', v')$ in *LS* if:

- A maximal set of m-labeled message occurrences are enabled at $(\bar{c}, v)$, and there is no chart L_i whose local configuration $((\bar{c})_i, v)$ will be hot-violated by an m-labeled message. In this case, for all charts L_j that have an m-labeled message occurrence enabled at $(\bar{c}, v)$, the $\xrightarrow{m}$ message synchronization steps will occur simultaneously; and

there is a silent step $(\bar{c}, v) \xrightarrow{\tau} (\bar{c}', v)$ in *LS* if:

- There is a chart L_i such that $((\bar{c})_i, v) \xrightarrow{\tau} ((\bar{c}')_i, v)$. In this case, for all $j \neq i$, we have $\bar{c}'_j = \bar{c}_j$; and

there is a time delay step $(\bar{c}, v) \xrightarrow{d} (\bar{c}, v + d)$ in *LS* if:

- For all $1 \leq i \leq n$, we have $((\bar{c})_i, v) \xrightarrow{d} ((\bar{c})_i, v + d)$. □

In the first case above, the *global condition* for all m-labeled message occurrences is the conjunction of all individual conditions, and the *global assignment* is the union of all individual assignments.

Similarly, we can define runs and $\rightarrow^{*}$ for a set of time-enriched LSC charts.

Definition 8 (satisfaction of an LSC system) A timed trace $\gamma \in (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{\omega}$ *satisfies* (passes) an LSC system *LS*, denoted $\gamma \models LS$, iff γ corresponds to an infinite run of *LS*, and it satisfies each chart L_i in *LS* separately. □

3.2.3 Semantics for existential charts

As mentioned in Section 1, unlike a universal chart which needs to be matched by each possible run of the system, an existential chart requires only one satisfying run of the system. Similar to Section 3.2.1, we can define the semantics for a single existential chart.

4 Verifying state/transition-based models against LSC requirements

A monitored LSC chart *L* can be used to specify a scenario-based user requirement on a state/transition-based real-time system model *S* (i.e., a network of timed automata). We may wish to model check *S* against *L*. However, there is no direct solution to this problem, because model checking by definition works only on state transition systems and temporal logic formulas.

To model check real-time systems against complex user requirements, a number of techniques which use (manually crafted) observer timed automata have been developed [1,19]. We notice that a monitored LSC chart in principle functions like an observer timed automaton — the chart keeps track of the progress of the system, and reports error once there is an unexpected event or timing error in the system. In order to make use of existing observer automata-based techniques and tools for scenario-based automatic verification, we need to automatically construct a behavior-equivalent observer timed automaton O_L for chart *L*, let O_L observe *S* in a non-intrusive way, and automatically extract a temporal logic formula which together with O_L characterizes the LSC requirement.

4.1 LSC-to-TA translation: one automaton per chart

4.1.1 Basic ideas of translation

By comparing the semantic states of a timed automaton and an LSC chart, we notice that they agree on the clock valuation part, but differ on the discrete part. In a timed automaton the control switches from one location to another, whereas in an LSC chart the control progresses from one cut to a next cut. Indeed an LSC cut is comparable to a TA location — if we view all the instances of an LSC chart collectively as a whole system (i.e., an automaton), then a cut can be viewed as a control "location" of this system. Based on the above consideration, it makes sense to translate an LSC cut to a TA location.

As mentioned in Sections 3.1.2 and 3.2.1, a simregion triggers the advancement from one LSC cut to a next cut. If we ignore the trivial cut advancement steps (i.e., the three latter cases of a "silent step"), then simregion is the only primitive semantic unit that can be stepped over to make a discrete (i.e., cut-changing) advancement step. This is very similar to a labeled edge in a timed automaton, which connects two TA locations. Therefore, it makes sense to translate an LSC simregion to a TA edge.

The basic rules for mapping live sequence charts to timed automata are given in Table 1, which will be explained in more detail later.

Table 1 Mapping live sequence charts to timed automata ("one-TA-per-chart")

	LSC		TA
explicitly specified behaviors	LSC cut	⇔	TA location
	$Mch.\top$	⇔	l_{min}
	$Mch.\bot$	⇔	l_{max}
	LSC simregion	⇔	TA edge
	message simregion	⇔	synchronization transition edge
	non-message simregion	⇔	silent transition edge
implicitly allowed behaviors	unconstrained event	⇔	self-loop edge
	cold violation	⇔	negated-guard transition edge + out-of-order sync. transition edge

For the example in Section 2, the original real-time system S consists of timed automata A, B, C and D, having channels m_1, m_2, m_3 and m_4, and clock variable x (Fig. 2); and the scenario-based LSC requirement L is given in Fig. 3. The translated observer timed automaton for chart L is presented in Fig. 7.

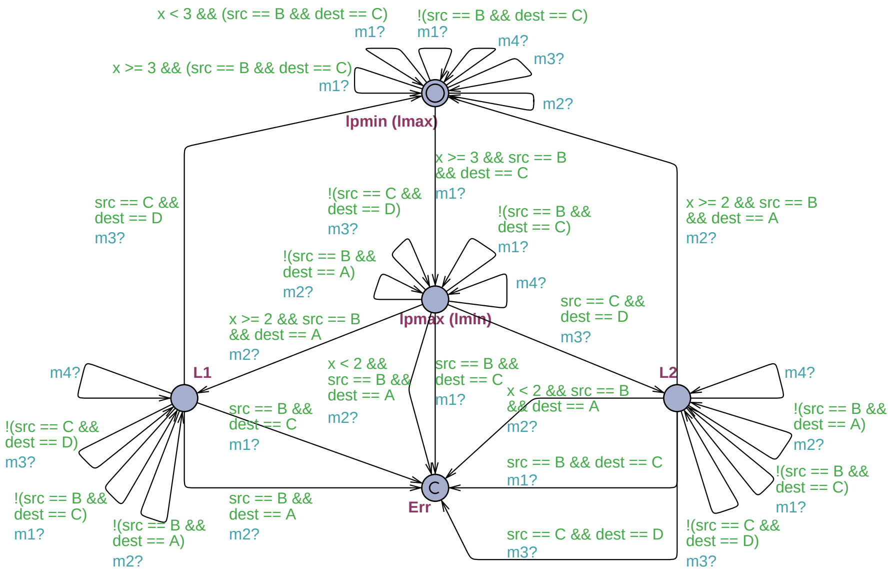


Fig. 7 The translated observer timed automaton O_L for chart L in Fig. 3 ("one-TA-per-chart" translation)

4.1.2 Translating an LSC cut to a TA location

The initial cut of an LSC chart is the minimal cut of the prechart $Pch.\top$. Starting from $Pch.\top$, the cut advances towards $Mch.\bot$ by stepping over a simregion, or by an intra-chart coordination, or by the autonomous advancement of an instance line.

For the minimal cut $Pch.\top$ and maximal cut $Pch.\bot$ of the prechart (resp. $Mch.\top$ and $Mch.\bot$ of main chart), we assign the TA locations l_{pmin} and l_{pmax} (resp. l_{min} and l_{max}, see Table 1), respectively. The l_{pmin}, l_{pmax}, l_{min} and l_{max} locations correspond to the four mandatory synchronization points for all instance lines in the chart.

Each time a simregion is stepped over, we create a new cut. In comparison, the cut advancement that is caused by intra-chart coordination or the autonomous advancement of an instance line has only a "gluing" or "managerial" semantics. In order not to clutter up the translated timed automaton, in the following cases we assign two adjacent cuts the same TA location in O_L:

- $Pch.\bot$ and $Mch.\top$; (i.e., l_{pmax} and l_{min} are the same location in O_L, meaning that the successful completion of prechart will immediately activate the main chart. See Fig. 7)
- $Mch.\bot$ and $Pch.\top$; (i.e., l_{max} and l_{pmin} are the same location in O_L, meaning that a next round of monitoring will begin immediately after the successful completion of the main chart. See Fig. 7)
- If in chart L, instance I_i has no more interactions with the other instance lines (i.e., $I_1,\ldots,I_{i-1},I_{i+1},\ldots,I_n$) in the prechart, then the two cuts with frontiers $(psn(loc(c)_{\langle I_1\rangle}),\ldots,psn(loc(c)_{\langle I_{i-1}\rangle}),\ (Pch\_bot_{L,I_i}-1), psn(loc(c)_{\langle I_{i+1}\rangle}),\ldots,$ $psn(loc(c)_{\langle I_n\rangle}))$ and $(psn(loc(c)_{\langle I_1\rangle}),\ldots,psn(loc(c)_{\langle I_{i-1}\rangle}), Pch\_bot_{L,I_i},$ $psn(loc(c)_{\langle I_{i+1}\rangle}),\ldots,psn(loc(c)_{\langle I_n\rangle}))$ will be assigned the same TA location. Similarly for the case in the main chart.

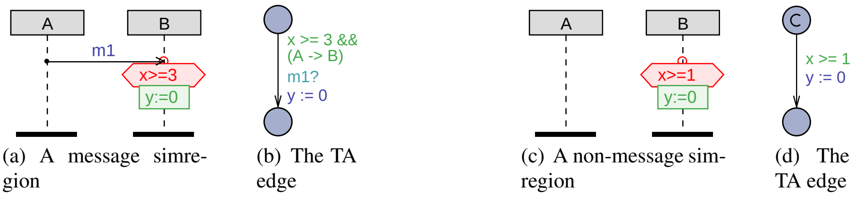
Since there are only finitely many instance lines and simregions in an LSC chart and there are no looping structures, the number of cuts will also be finitely many.

4.1.3 Translating an LSC simregion to a TA edge

If $s \in SR$ is a message simregion (Fig. 8(a), $(x \geq 3, m_1, y := 0)$), then we map the message, condition (if any) and assignment (if any) of s into one edge of the TA O_L (Fig. 8(b)).

In an LSC chart, a message in Σ is sent from one particular instance to another one (e.g., from instance A to B). To preserve this sender/receiver information in the translated timed automaton, the TA edge will be further guarded by the predicate "$A \dashrightarrow B$" (Fig. 8(b)), which is a shorthand for "$(src = A) \wedge (dest = B)$".

If $s \in SR$ is a non-message simregion (Fig. 8(c), $(x \geq 1, \varepsilon, y := 0)$), then according to the ASAP semantics, this simregion should be stepped over immediately. Since in UPPAAL timed automata, a *committed* location requires that it be exited immediately, we mark the source location of the translated TA edge as a committed location (Fig. 8(d), "C"-marked location).



(a) A message simregion (b) The TA edge (c) A non-message simregion (d) The TA edge

Fig. 8 Translating LSC simregion to TA edge (here "$A \rightarrow B$" denotes that instances A and B are the message sender and receiver, respectively)

4.1.4 Incremental construction of the TA

In Sections 4.1.2 and 4.1.3 we showed how to translate a single cut and simregion to a TA location and edge, respectively. When translating an entire chart into a timed automaton, we need to exploit the structural information of LSC to conduct incremental translation.

As mentioned in Section 3.1.2, simregion is the primitive semantic unit for triggering cut advancements. At a given cut there might be more than one next simregion that can be stepped over. Each simregion should correspond to an outgoing edge from the TA location that corresponds to the given cut. All of these simregions should be translated to TA edges that share the same common "source cut." But before we can do this, we should determine how all the simregions are partially ordered in a chart, and how these simregions and the cuts are partially ordered.

In Section 3.2 we showed how to determine the preorder relation $\preccurlyeq$ on the set $loc(L)$ of locations of chart L. Now the relation $\preccurlyeq$ on $loc(L)$ can be lifted up to a partial order $\preccurlyeq'$ on $(SR \cup Cuts)$ as follows:

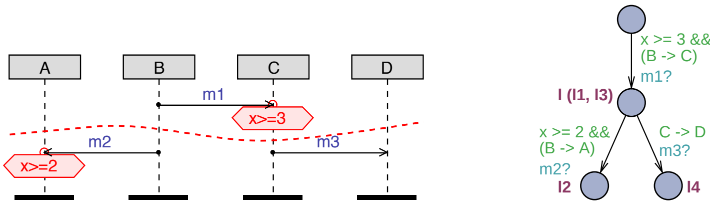
$$\forall s_1, s_2 \in (SR \cup Cuts) \,.\, (s_1 \preccurlyeq' s_2 \Leftrightarrow \exists l_1 \in loc(s_1), l_2 \in loc(s_2) \,.\, l_1 \preccurlyeq l_2).$$

For instance in Fig. 3, the partial order $\preccurlyeq'$ among the three simregions s_1 (middle), s_2 (left) and s_3 (right) is: $s_1 \preccurlyeq' s_2$, and $s_1 \preccurlyeq' s_3$.

Assume that a TA location l has already been created for the current LSC cut (Fig. 9(a), cut $\{s_1 = (x \geq 3, m_1, \varepsilon)\}$, and Fig. 9(b), TA location l). Without loss of generality, we assume that there are two immediately following simregions s_2 and s_3.

If s_2 and s_3 are both message simregions (Fig. 9(a)), then the two new TA edges will be appended to location l. The LSC and TA semantics coincide that there are two possible orderings of the execution of the two transitions. Let the two new edges be (l_1, l_2) and (l_3, l_4), respectively. Then l_1 and l_3 will be superposed on l (Fig. 9(b)). The cut will be advanced accordingly.

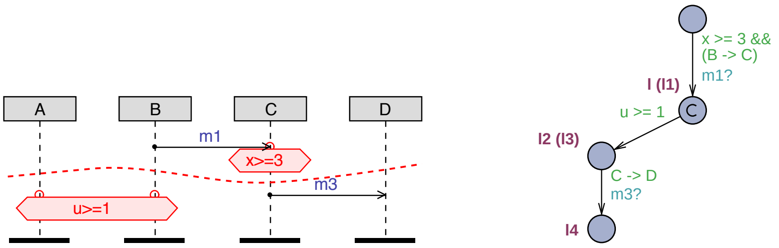
If s_2 is a non-message simregion that spans across instance lines A and B, e.g., $(u \geq 1, \varepsilon, \varepsilon)$ (Fig. 10(a)), then according to the ASAP semantics, s_2 will be stepped over immediately, and s_3 will follow, but cannot be the other way around. When appending the corresponding two edges (l_1, l_2) and (l_3, l_4) to the TA, we mark the source location l_1 as a committed location, and superpose it on l (Fig. 10(b), edge (l_1, l_2)). Note that in this case l_2 may not be superposed on l. Therefore, there is only one possible ordering of edges (l_1, l_2) and (l_3, l_4) (Fig. 10(b)).



(a) LSC cut with subsequent simregions $s_2 = (x \geq 2, m_2, \varepsilon)$ and $s_3 = (\varepsilon, m_3, \varepsilon)$

(b) Translated TA fragment

Fig. 9 TA edge construction for two subsequent message simregions

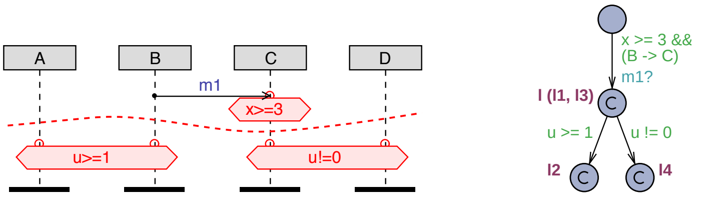


(a) LSC cut with subsequent simregions $s_2 = (u \geq 1, \varepsilon, \varepsilon)$ and $s_3 = (\varepsilon, m_3, \varepsilon)$

(b) Translated TA fragment

Fig. 10 TA edge construction for a subsequent message and non-message simregions

If s_2 and s_3 are both non-message simregions, e.g., $(u \geq 1, \varepsilon, \varepsilon)$ and $(u \neq 0, \varepsilon, \varepsilon)$ (Fig. 11(a)), then according to the ASAP semantics, both (l_1, l_2) and (l_3, l_4) will be executed immediately, therefore there are again two possible orderings of the execution of the two transitions (Fig. 11(b)).



(a) LSC cut with subsequent simregions $s_2 = (u \geq 1, \varepsilon, \varepsilon)$ and $s_3 = (u \neq 0, \varepsilon, \varepsilon)$

(b) Translated TA fragment

Fig. 11 TA edge construction for two subsequent non-message simregions

4.1.5 Implicitly allowed behavior

In addition to the *explicitly* specified behaviors, an LSC chart also has behaviors that are *implicitly* allowed, e.g., those due to unconstrained events and cold violations (Table 1).

Let Π correspond to the set of possible messages in the system model S (i.e., a network of timed automata), and $\Sigma \subseteq \Pi$ be the set of messages in chart L. Clearly, messages in $(\Pi \backslash \Sigma)$ are not constrained by chart L. For each message in $(\Pi \backslash \Sigma)$, we add a corresponding self-loop edge to each non-committed location l of the translated TA O_L. For example in Fig. 7, at location l_{pmin} the unconstrained events correspond to the edges labeled with m_4, $[!(src == B \ \&\& \ dest == C)]m_1$, $[!(src == B \ \&\& \ dest == A)]m_2$ and $[!(src == C \ \&\& \ dest == D)]m_3$. Note that the latter two edges will be merged with some cold violation edges that will be explained shortly, thus giving rise to the m_2- and m_3-labeled edges, respectively.

According to the LSC semantics, cold violations in the prechart or main chart are not failures. Rather they just bring the chart back to the prechart minimal cut. To model this, for a cut c and each following simregion s that has a cold condition g, we add edges from the corresponding TA location l to l_{pmin} to correspond to the $\neg g$ conditions (of DNF form) (Fig. 7, the edge labeled with $[(x < 3) \ \&\& \ (src == B \ \&\& \ dest == C)]m_1$ from l_{pmin} to l_{pmin}). Similarly, given a cut c in the prechart, for each m-labeled message that occurs in L but does not follow c immediately (i.e., it violates the event partial order), we also add a corresponding TA edge (l, l_{pmin}) (Fig. 7, the edges labeled with $[(src == B \ \&\& \ dest == A)]m_2$ and $[(src == C \ \&\& \ dest == D)]m_3$ from l_{pmin} to l_{pmin}).

4.1.6 Undesired behavior

The construction of the observer TA so far considers only the legal (or admissible) behaviors. When the current configuration (c, v) is in the main chart, if an observed message is not enabled at cut c (i.e., it is an out-of-order message), or the hot condition of the simregion that immediately follows c evaluates to false under v, then there will be a hot violation. In this case, we add a dead-end (sink) location Err in the TA, and for each such hot violation we add an edge to Err (Fig. 7).

4.1.7 Invariant activation mode

According to the LSC semantics, under the invariant mode the prechart is being continuously monitored. Normally we need to maintain multiple incarnations of the chart. In this way, a given message sequence will not be incorrectly rejected by the chart (i.e., there is no false negative). For instance, given the chart in Fig. 3, and given a message sequence $m_1 \cdot m_1 \cdot m_2 \cdot m_3$ (assuming that the message guards in Fig. 3 are all satisfied), although the first incarnation of the chart hot-violates this sequence (i.e., the second m_1 violates the message partial order in the main chart), the second incarnation works well with it (i.e., the latter three messages $m_1 \cdot m_2 \cdot m_3$ match the chart).

To enforce the LSC semantics under the invariant activation mode, for each message occurrence that appears in *Pch* as a minimal event (i.e., an event that is minimal in the event partial order induced by the chart), we add a corresponding self-loop to location l_{pmin} (Fig. 7, the $[(x >= 3) \ \&\& \ (src == B \ \&\& \ dest == C)]m_1$-labeled self-loop edge at l_{pmin}). We call this kind of handling *prechart pre-matching*.

4.2 Complexity

Let the number of simregions that appears in L be n. In the worst case, the number of locations in the translated observer TA O_L is $2^n + 1$. We have this exponential complexity when L consists of only the prechart or the main chart, and the messages in L are totally unordered.

The number of outgoing edges from a location l of O_L depends on: (1) the number of unconstrained events, *ue*; (2) the number of the following simregions in the corresponding cut c of L, fs; (3) the length of the condition (in case the condition evaluates to false), lc; and (4) the number of messages that cause violations of the chart, cv. Therefore, the number of outgoing edges from a TA location has complexity $O(ue + fs + lc + cv)$.

Since the LSC simregions are directly copied as TA edges, O_L has the same numbers of synchronization channels (message labels) and clock variables as L.

4.3 Equivalence of LSC and TA

If in the translated timed automaton O_L of chart L we ignore the undesired and implicitly allowed behaviors, i.e., we ignore the edges that correspond to hot violations, unconstrained events, cold violations, and prechart pre-matching, then we have:

Theorem 1 *If a configuration (c,v) of L corresponds to a semantic state (l,v) of O_L, then: (1) each simregion s that follows (c,v) in L uniquely corresponds to an outgoing edge (l,l') in O_L; and (2) the target configuration (c',v') of s in L uniquely corresponds to the target semantic state (l',v') in O_L.* □

Theorem 1 says that each LSC configuration is uniquely mapped to a semantic state of the observer timed automaton, and each LSC simregion is uniquely mapped to an edge of that automaton.

If in the LSC semantics (Section 3.2.1) we ignore the silent steps that are caused by intra-chart coordinations and autonomous advancements of instance lines, then based on Theorem 1, we have the following theorem:

Theorem 2 *For any trace tr in O_L: $tr \models L \Leftrightarrow (O_L, tr) \models (l_{min} \leadsto l_{max})$.* □

Theorem 2 says that a trace in the observer timed automaton of an LSC chart satisfies that chart if and only if the trace satisfies the "leads-to" property $(l_{min} \leadsto l_{max})$ in that automaton. Alternatively, it indicates that O_L has exactly the same set of *legal* traces as L.

As we can anticipate, the prechart pre-matching mechanism in Section 4.1.7 does introduce undesired behaviors and non-determinacy. For instance in Fig. 7, the message sequence $tr = m_1 \cdot m_1 \cdot m_2 \cdot m_3$ could be an accepted trace in O_L (assuming that all message guards are satisfied). But since its sub-sequence $tr' = m_1 \cdot m_1$ can be rejected (i.e., leading to a hot violation), so tr does not really satisfy L. However, it coincides that this particular trace tr in the model O_L does not satisfy the CTL property $(l_{min} \leadsto l_{max})$ as well.

The syntax and semantics of timed automata are given in Appendix B. The proofs of theorems in Section 4 and the lemma for them can be found in Appendix C.

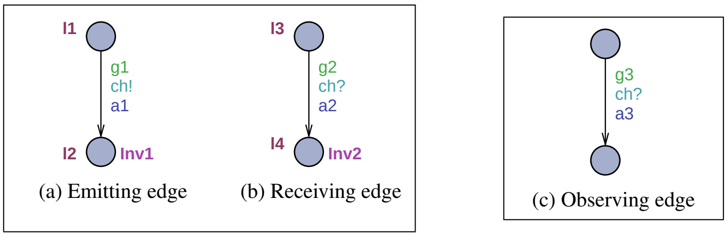
4.4 Composing the observer automaton with the original system

With the development so far in this section (Section 4), the set X of clocks can be viewed as "private" clocks of the LSC chart L, i.e., L can both read and *reset* these clocks. When we use the observer timed automaton O_L to observe the original system S, then the set C_S of clocks in S will also be visible to (but cannot be reset by) O_L. To this end, the definition of the clock valuation part of a configuration of L will be extended accordingly.

When composing O_L with S, we wish that O_L would "observe" S in a *timely* and *non-intrusive* manner. A natural idea is to let the synchronization channels in O_L (and accordingly the relevant channels in S) be broadcast channels to achieve this goal. However, this is not possible because UPPAAL has a restriction that broadcast channels cannot be guarded by timing constraints. To solve this problem, we propose to use spying techniques such that the translated observer TA will be notified of each message synchronization in the original system *immediately after* it occurs there. Specifically, for each channel $ch \in \Pi$, we make the following modifications:

(1) In S (e.g., Fig. 12(a)-12(b)), for each edge (l_1, l_2) that is labeled with ch!, we add an intermediate committed location l'_1 and a cho!-labeled edge in between edge (l_1, l_2) and location l_2. Here cho is a dedicated fresh channel which aims to notify O_L of the **o**ccurrence of the ch-synchronization in S. The location invariant (if any) of l_2 will be copied on to l'_1. Furthermore, we use a global boolean flag variable (i.e., a binary semaphore) *mayFire* to further guard the ch-synchronization. This semaphore is initialized to true at system start. It is cleared immediately after the ch-synchronization in S is taken, and it is set again immediately after the cho-synchronization is taken (Fig. 13(a)).

(2) In O_L (e.g., Fig. 12(c)), each synchronization label ch? is renamed to cho? (Fig. 13(b)).



(a) Emitting edge

(b) Receiving edge

(c) Observing edge

Fig. 12 Coupling emitting/receiving edges in original system model S and the observing edge in observer timed automaton O_L

If L has non-message simregions, then O_L contains corresponding committed locations. If in a certain state both O_L and some timed automata in S are in committed locations (e.g., l_{m+1} in Fig. 14(b), and l_2 in Fig. 14(a)), there will be a race condition.

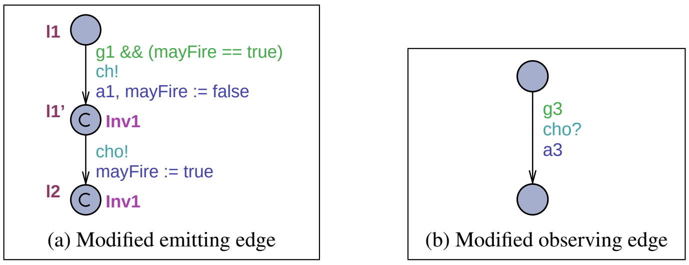


Fig. 13 Edge modifications for Fig. 12

But according to the ASAP semantics of L, the (internal action) edge in O_L has higher priority. To this end, for each edge (l_i, l_{i+1}) in O_L, if l_{i+1} is a committed location, then we add "*NxtCmt* := true" to the assignment of the edge, otherwise we add "*NxtCmt* := false". Here the global boolean flag variable (i.e., a binary semaphore) *NxtCmt* denotes whether the observer TA will be in a committed location (Fig. 15(b)). This semaphore is initialized to false at system start. Accordingly, for each ch-labeled edge (l_i, l_{i+1}) in S where $ch \in \Pi$ and l_i is a committed location, we use "*NxtCmt* == false" to strengthen the condition of the edge (Fig. 15(a)).

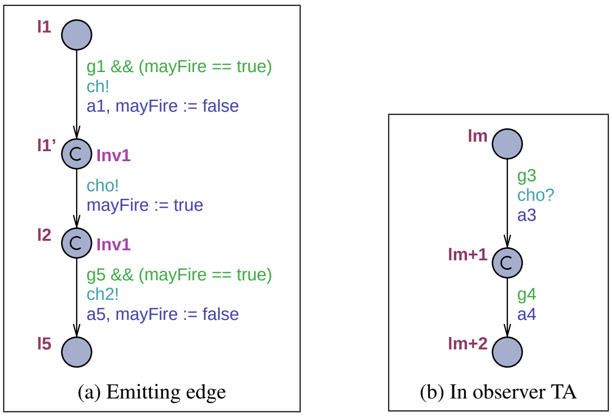
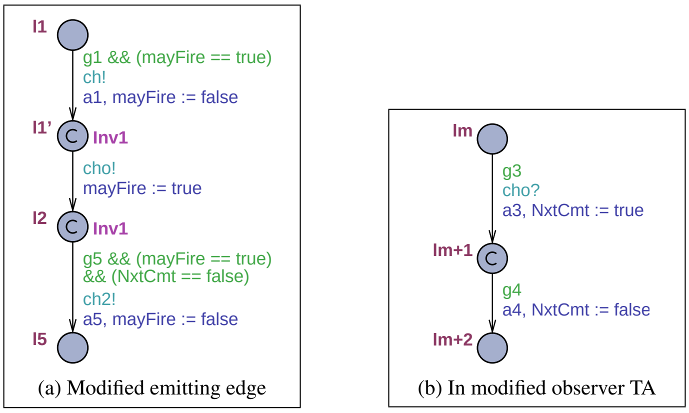


Fig. 14 Race condition for Fig. 13 due to a non-message simregion in LSC chart (and thus a committed location l_{m+1} in observer timed automaton)

Our method of composing the observer timed automaton O_L with the original system model S is similar to that of [14]. While their method works only when the target state of a communication action is not a committed location in the original model, in our method, due to the first locking mechanism (using *mayFire*), we have no restrictions on whether a location in S is a normal, urgent or committed one. Broadcast channels can be handled the same way as binary synchronization channels in our method. Furthermore, due to the second locking mechanism (using *NxtCmt*), we guarantee the enforcement of the ASAP semantics in O_L.



(a) Modified emitting edge

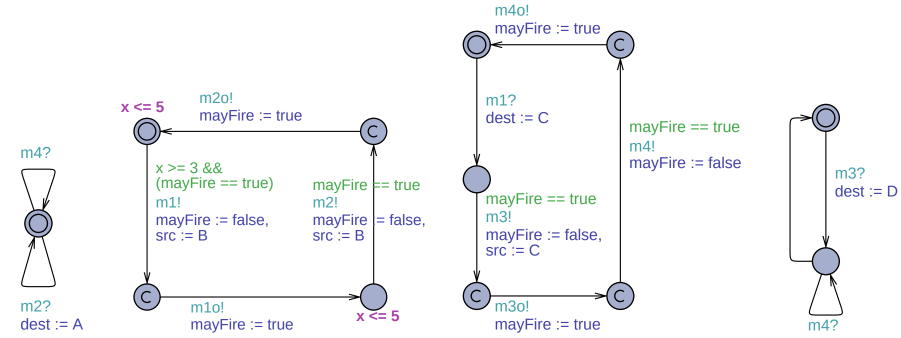
(b) In modified observer TA

Fig. 15 Further modifications for Fig. 14

Our method involves only syntactic scanning and manipulations. For each $ch \in \Pi$, we need to introduce a dedicated fresh channel cho. For each occurrence of the emitting edge ch!, we need to introduce a fresh committed location in S. Moreover, we need two global boolean flag variables (*mayFire*, *NxtCmt*) as the binary semaphores.

Example of Section 2 Continued

After modifying S and the automatically generated observer TA O_L, we get the composed network of instrumented TAs $(S' \,||\, O'_L)$ (Fig. 16 and 17). Note that in Fig. 17 each channel name has been suffixed with an "o", e.g., m1 becomes m1o.



(a) TA A' (b) TA B' (c) TA C' (d) TA D'

Fig. 16 The modified model S' of the original real-time system in Fig. 2(a)-2(d)

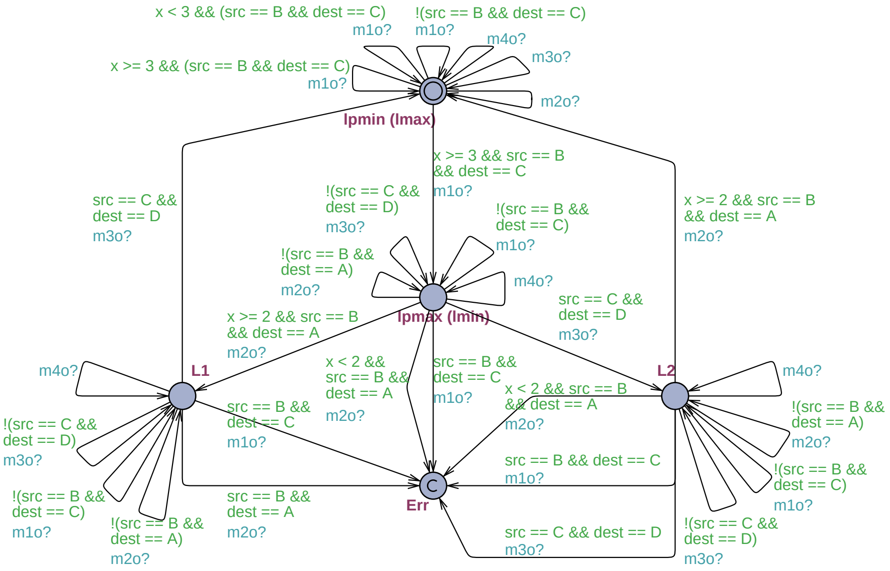


Fig. 17 The modified version of the translated observer timed automaton O'_L (Fig. 7) of the chart in Fig. 3

4.5 Verification problems

After the modifications, the original system model S becomes S', and the observer timed automaton O_L for chart L becomes O'_L. Let the minimal and maximal cuts of the main chart of L correspond to locations l_{min} and l_{max} of O'_L, respectively. Recall that the UPPAAL “leads-to” property $(\phi \leadsto \varphi)$ stands for $\mathsf{A}\Box(\phi \Rightarrow \mathsf{A}\Diamond\,\varphi)$, where ϕ and φ are state formulas.

When L is a universal chart, we have the following main theorem:

Theorem 3 $S \models L \Leftrightarrow (S' \,||\, O'_L) \models (l_{min} \leadsto l_{max})$. □

Theorem 3 says that a TA-modeled real-time system S satisfies a universal LSC chart requirement L if and only if the parallel composition of the (instrumented) system and the observer timed automaton satisfies the “leads-to” property $(l_{min} \leadsto l_{max})$. This indicates that the problem of model checking real-time systems against universal LSC chart requirements can be equivalently transformed into a CTL real-time model checking problem in UPPAAL.

When L is an existential chart, we have another main theorem:

Theorem 4 $S \models L \Leftrightarrow (S' \,||\, O'_L) \models \mathsf{E}\Diamond\, l_{max}$. □

Theorem 4 says that a TA-modeled real-time system S satisfies an existential LSC chart requirement L if and only if the parallel composition of the (instrumented) system and the observer timed automaton satisfies the reachability property $\mathsf{E}\Diamond\, l_{max}$.

Furthermore, it is possible to check whether a system S satisfies multiple existential charts $L_1, L_2, \ldots, L_m$ *simultaneously* by checking the formula $\mathsf{E}\Diamond\, (l_{1,max} \wedge l_{2,max} \wedge$

$\ldots \wedge l_{m,max})$, where $l_{i,max}$ denotes the location in observer TA O'_{L_i} that corresponds to the maximal cut of the main chart of existential chart L_i, $1 \leq i \leq m$.

Example of Section 2 Continued

For the composed network of instrumented timed automata in Fig. 16 and 17, we check in UPPAAL the property $(l_{min} \rightsquigarrow l_{max})$, and it turns out to be satisfied. This indicates that S does satisfy the requirements that are specified in L.

If in L the condition of m_2 is changed from $x \geq 2$ to e.g. $x \geq 4$, then the property is found not satisfied. There will be a counterexample, e.g., when O'_L gets to location `L2` (Fig. 17), but the value of clock x falls in $[3,4)$, then it will get stuck in location `Err` and will never be able to arrive at location l_{max} thereafter.

5 Verifying object interaction-based models against LSC requirements

In addition to being used as a property specification language, LSC can also be used as a scenario-based behavioral modeling language (i.e., as a high-level "programming" language [18]). When some LSC charts are used for behavioral modeling and some others are used for property specification, it will be possible to verify scenario-based LSC models against scenario-based LSC requirements. Clearly, this contributes to earlier validation of the prototyped systems.

5.1 LSC-to-TA translation: one automaton per instance line

5.1.1 Motivation

Similar to Section 4, in this section our method of verifying object interaction-based system models against LSC requirements relies on a translation of the LSC charts to timed automata, and the reduction of the verification problems to `CTL` real-time model checking problems.

However, unlike monitored charts which each specify a piece of user requirements individually, a set of driving charts are supposed to characterize the inter-object behaviors of the system *collectively*. When the system consists of a large number of driving charts, then the cut-based LSC-to-TA translation will encounter the state explosion problem: the number of possible global cuts (i.e., the number of possible system states) will increase rapidly, and explicit encoding and storing these information requires a lot of memory. Furthermore, the outcome of the translation as a single huge timed automaton will be difficult to visualize, to debug and to diagnose.

To overcome the above problems, in this section we propose a different method for translating LSC charts into timed automata. For each driving LSC chart L in the system model, we view the instance lines in L as a set of parallel composed processes that communicate with one another and collaborate to achieve a common goal as specified by chart L. Since UPPAAL also operates on a network of parallelly composed processes (TAs) that communicate with each other, this motivates us to translate each instance line of L to a timed automaton. In this way we avoid the explicit construction of a global automaton. This idea in spirit resembles the approaches of [17,37], which aim at smart play-out and satisfiability checking, respectively; and it is also similar to the approaches of mapping message-based concurrent objects to TAs [22,21], by means of which modular schedulability analysis of distributed real-time systems can be achieved. Thanks to the UPPAAL features of broadcast channels, boolean and integer variables and committed locations in timed automata, we are able to appropriately translate the LSC features such as intra/inter-chart coordinations and cold/hot violations to timed automata. Compared with the "one-TA-per-chart" approach that can be viewed as a kind of *centralized* translation (Section 4.1), the "one-TA-per-instance line" approach of this section can be viewed as a kind of *distributed* translation.

Table 2 gives an overview of the most important LSC-to-TA mapping rules. These rules will be explained in more detail later.

Table 2 Mapping live sequence charts to timed automata ("one-TA-per-instance line")

	LSC		TA
basic mapping rules	chart L_u	$\Longleftrightarrow$	TAs $A_{u,1} \Vert A_{u,2} \Vert \ldots \Vert A_{u,inst(L_u)} \Vert Coord_u$
	instance line $I_i,\ 1 \leq i \leq inst(L_u)$	$\Longleftrightarrow$	TA $A_{u,i}$
	position $k,\ 0 \leq k \leq p\_max_{L_u,I_i}$	$\overset{\text{R1}}{\Longleftrightarrow}$	location l_k
	message I_i (at position k) $\xrightarrow{m} I_j$	$\overset{\text{R2}}{\Longleftrightarrow}$	edge $l_{k-1} \xrightarrow{m!} l_k$
	message I_i (at position k) $\xleftarrow{m} I_j$	$\overset{\text{R2}}{\Longleftrightarrow}$	edge $l_{k-1} \xrightarrow{m?} l_k$
intra-chart coordination	chart L_u	$\Longrightarrow$	coordinator automaton $Coord_u$
	instance line I_i progressing to its		
	$Pch\_bot_{L_u,I_i}$ position	$\overset{\text{R3}}{\Longleftrightarrow}$	$pch\_over_{u,i}$-sync. from $A_{u,i}$ to $Coord_u$
	$Mch\_bot_{L_u,I_i}$ position	$\overset{\text{R3}}{\Longleftrightarrow}$	$mch\_over_{u,i}$-sync. from $A_{u,i}$ to $Coord_u$
	$Pch\_bot_{L_u,I_i} \rightarrow Mch\_top_{L_u,I_i}$ (chart activation)	$\overset{\text{R4}}{\Longleftrightarrow}$	$activate_u$-broadcast sync. from $Coord_u$
	$Mch\_bot_{L_u,I_i} \rightarrow Pch\_top_{L_u,I_i}$ ("all over again")	$\overset{\text{R4}}{\Longleftrightarrow}$	$over_u$-broadcast sync. from $Coord_u$
inter-chart coordination	chart $L_u,\ I_i$ (at position k) $\xrightarrow{m} I_j$; and	$\overset{\text{R5}}{\Longleftrightarrow}$	accompany $(l_{k-1} \xrightarrow{m!} l_k)$ with $(l_{k-1} \xrightarrow{m?} l_k)$;
	chart $L_v\ (v \neq u),\ I_i$ (at position k') $\xrightarrow{m} I_j$		accompany $(l_{k'-1} \xrightarrow{m!} l_{k'})$ with $(l_{k'-1} \xrightarrow{m?} l_{k'})$
cold/hot violation	chart L_u		
	prechart violation due to out-of-order message on instance line I_i	$\overset{\text{R6}}{\Longleftrightarrow}$	pch_vio_u-sync. from $A_{u,i}$ to $Coord_u$, followed by $reset_u$-broadcast sync. from $Coord_u$
	main chart violation due to out-of-order message on instance line I_i	$\overset{\text{R7}}{\Longleftrightarrow}$	$A_{u,i}$ arriving at a deadend location
clock constraint	chart L_u		
	$I_i \xrightarrow{[g]\,m} I_j$, where g is a guard	$\overset{\text{R9}}{\Longleftrightarrow}$	message-emitting edge $\xrightarrow{m!}$ in $A_{u,i}$ with immediately prefixed/suffixed transitions that test the upper/lower bounds of g
clock reset	chart L_u		
	$I_i \xrightarrow{m\,/\,a} I_j$, here a is a set of clock resets and optionally chart L_v	$\overset{\text{R10}}{\Longleftrightarrow}$	$A_{u,i}$ sends an individual m_Rpt request for clock resets to dedicated timed automaton
	$I_i \xrightarrow{m\,/\,a'} I_j$, here a' is another set of clock resets		A_m, and A_m initiates system-wide m_Rst broadcast synchronization for clock resets

In order to keep the driving LSC charts and their complexity within human readable and manageable levels, a single LSC chart must not be very large and complex. Rather the complexity of scenario-based models mainly lies in the interplays of a large number of simple charts. Our LSC-to-TA translation is in accordance with this philosophy. Instead of constructing a complex global state machine that handles all possible activities explicitly, we leave the intricate semantics of LSC chart progress and intra/inter-chart coordinations mostly up to UPPAAL.

5.1.2 Translating message-only charts

As mentioned in Section 3.1, along each instance line I_i in chart L, there is a set $pos(L,I_i)$ of positions, among which there is a set $StdPos(L,I_i) \subset pos(L,I_i)$ of four "standard" positions. For example in instance line A of Fig. 4, there are 7 positions (black filled circles), where the four standard ones are $Pch\_top_{L,A}\,(0), Pch\_bot_{L,A}\,(3)$, $Mch\_top_{L,A}\,(4)$ and $Mch\_bot_{L,A}\,(6)$.

Fig. 18 shows the translated network of timed automata for the chart L_1 in Fig. 4.

(1) Basic mapping rules

Let *LS* be an LSC system, L_u be a chart in *LS*, and I_i be an instance line in L_u. We map each such I_i to a timed automaton $A_{u,i}$ using the following rules (Table 2, row 1 "basic mapping rules"):

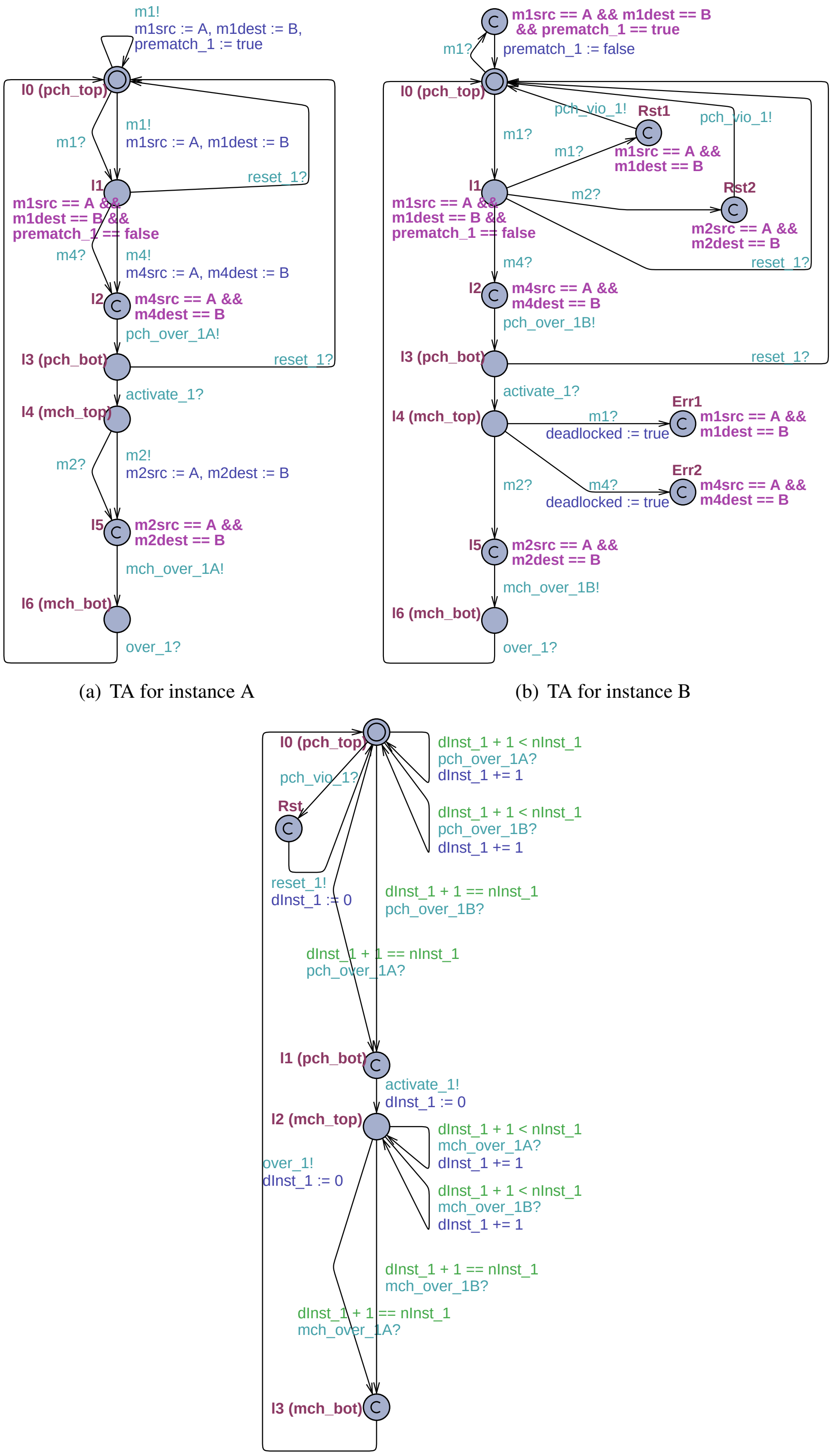
- R1 Each position k on I_i of L_u corresponds to a TA location l_k in $A_{u,i}$, $0 \leq k \leq p\_max_{L_u,I_i}$. See Fig. 18(a), locations l_0 - l_6.
- R2 If at position k on I_i of L_u there is a sending of an m-labeled message to instance I_j, then an $m!$-labeled TA edge from location l_{k-1} to l_k will be assigned in $A_{u,i}$. See Fig. 18(a), straight line edges $(l_0, l_1), (l_1, l_2), (l_4, l_5)$. Similarly, if it is a message receiving, then an $m?$-labeled TA edge will be assigned. See Fig. 18(b), straight line edges $(l_0, l_1), (l_1, l_2), (l_4, l_5)$.

We use the notations $Pch\_top, Pch\_bot, Mch\_top$ and Mch_bot to also denote the TA locations that correspond to their respective LSC positions on the instance line. For any position k other than the aforementioned four, it corresponds to a TA location l_k, meaning that upon sending/receiving the message that anchors at position k, we now arrive at l_k. Specifically, position 0 (i.e., *Pch_top*) corresponds to the initial TA location l_0 (i.e., the "abused" *Pch_top*).

When the message sending case of rule R2 is applied, the message-emitting TA edge can be associated with an assignment "$m\_src := I_i$, $m\_dest := I_j$", where *m_src* and *m_dest* are fresh auxiliary (bounded integer) variables, meaning that an m-labeled message is sent from instance I_i to I_j in chart L_u. In R2, the destination location l_k will have invariant "$(m\_src == I_i) \wedge (m\_dest == I_j)$" and "$(m\_src == I_j) \wedge (m\_dest == I_i)$" for message sending and receiving, respectively (Fig. 18(a), locations l_1, l_2, l_5, and Fig. 18(b), l_1, l_2, l_5).

(2) Handling intra-chart coordinations

In an LSC chart, if an instance line (process) in its prechart portion has no more interactions with the other instance lines (e.g., it has successfully sent/received the



(a) TA for instance A

(b) TA for instance B

(c) The coordinator automaton

Fig. 18 The translated network of timed automata for the untimed chart in Fig. 4 ("one-TA-per-instance line" translation)

last message, or it does not really span across the prechart and thus before chart activation it has no interactions with other instance lines at all), then it will *immediately* progress to the bottom position *Pch_bot* of its prechart portion, to be ready for a next mandatory synchronization that involves all the instance lines in that chart (Table 2, row 2 “intra-chart coordination”, upper-middle part).

R3 At position k on the prechart portion of I_i of L_u, if $k = Pch\_bot_{L_u,I_i} - 1$, then we mark l_k as a committed location in $A_{u,i}$, and we add a $pch\_over_{u,i}!$-labeled edge from l_k to l_{k+1} (i.e., $Pch\_bot_{L_u,I_i}$) in $A_{u,i}$. See Fig. 18(a), location l_2.

The auxiliary channel $pch\_over_{u,i}$ is used to notify the coordinator automaton $Coord_u$ (explained below) of the completion of instance line I_i with its prechart portion in chart L_u.

When all the instance lines in chart L_u progress to their respective *Pch_bot* positions, the prechart is successfully matched. Once this happens, all these instance lines must immediately synchronize and progress to their respective *Mch_top* positions, meaning that the main chart is now activated. To model this kind of *intra-chart* coordination at the prechart/main chart interface, for each chart L_u, we create a dedicated (auxiliary) coordinator automaton $Coord_u$ (note that it is an untimed automaton except for the notion of time that is induced by the UPPAAL committed locations). This automaton will communicate with the automata that correspond to the instance lines of L_u using auxiliary binary channels such that it can bookkeep how many instance lines are done with their prechart portions. Once the coordinator automaton realizes that the prechart has been successfully matched, it will immediately launch a broadcast synchronization with all the timed automata that correspond to the instance lines by using a broadcast channel.

Fig. 18(c) gives an example of the coordinator automaton for chart L_1 of Fig. 4, where $pch\_over_{1,A}$ and $pch\_over_{1,B}$ are binary channels, $activate_1$ is a broadcast channel which indicates that the main chart is to be activated, $nInst_1$ is a constant that denotes the <u>n</u>umber of <u>inst</u>ance lines that participate in chart L_1 (hence “*nInst*”), and $dInst_1$ is an integer variable that denotes the number of <u>inst</u>ance lines that are <u>d</u>one with their prechart (or main chart) portions of L_1 (hence “*dInst*”).

The coordinator automaton synchronizes all the timed automata that correspond to the instance lines in the chart for a collective advancement (i.e., one onward advancement step for each automaton) according to the following rule (Table 2, row 2, lower part):

R4 At position k of I_i of L_u, if $k = Pch\_bot_{L_u,I_i}$, then an $activate_u?$-labeled TA edge from l_k to l_{k+1} will be assigned in $A_{u,i}$. See Fig. 18(a), l_3, and Fig. 18(b), l_3.

Similarly, intra-chart coordination upon main chart completion will correspond to the channels $mch\_over_{u,i}$ (“instance line I_i of chart L_u has completed its main chart portion”) and $over_u$ (“now that the main chart has been successfully matched, the matching process of chart L_u will start all over again”).

(3) Handling inter-chart coordinations

In scenario-based modeling, the same message may well appear in two or more charts. For example given an LSC system *LS*, in chart L_1 there is an m-labeled message occurrence mo_1 sent from instance I_1 to I_2, and in chart L_2 there is an m-labeled message occurrence mo_2, also from I_1 to I_2. If at a certain global configuration $(\bar{c}, v)$ these message occurrences (in the above example mo_1 and mo_2) are all enabled, then their firings should be synchronized, i.e., either all of them are chosen to be fired, or none of them is chosen. This is considered a kind of *inter-chart* coordination.

In the translated network of timed automata, this coordination can be accomplished by using a broadcast synchronization (in the above example we let m be a broadcast channel). Recall that in a broadcast synchronization, there is only one sender. Therefore, when translating the message occurrences (in the above example mo_1 and mo_2) to edges in their respective timed automata, only one of the LSC positions that are associated with the message tails (i.e., sending locations) in LS can correspond to the sole message-emitting TA edge in the translated TAs, and all others will correspond to message-receiving TA edges. But considering that all message-sending instance lines in the relevant charts have the equal possibility to initiate the message broadcast synchronization, we propose a universal and symmetric solution: for each $m!$-labeled edge from one TA location to another, we add an $m?$-labeled edge between these two TA locations to "accompany" the $m!$-labeled edge. In other words, we let all translated TA locations that correspond to the message-sending locations (in the above example two locations in the translated TAs $A_{1,1}$ and $A_{2,1}$) have the equal chance to act as the broadcast synchronization initiator (Table 2, row 3).

R5 If at position k on I_i of L_u there is a sending of an m-labeled message, then an $m?$-labeled TA edge from l_{k-1} to l_k will be added in $A_{u,i}$. In the translated TAs, m will be changed from a binary to a broadcast channel. See Fig. 18(a), polyline edges (l_0, l_1), (l_1, l_2), (l_4, l_5).

(4) Handling cold and hot violations

Along an instance line of an LSC chart, if an arriving message is not enabled at the current cut in the prechart, then there will be a cold violation. In this case, all participating instance lines in this chart should be reset (i.e., brought back to their initial positions) immediately. In the translation, this is accomplished by letting the timed automaton that corresponds to the message receiving instance line "report" the cold violation to the coordinator automaton in charge, which in turn immediately initiates a broadcast synchronization to ask the timed automata that correspond to all other instance lines of the chart to do a reset (Table 2, row 4, upper part).

R6 Assume that at position k on the prechart portion of instance line I_i of chart L_u, there is a receiving of an m-labeled message from instance I_j. If $k \geq Pch\_top_{L_u,I_i} + 2$, then for all m'-labeled messages in L_u such that $m' \neq m$ (note that $m, m' \in \Pi$), first an $m'?$-labeled outgoing TA edge from l_{k-1} will be added, then a fresh intermediate committed TA location with invariant $m'\_src == src(m') \wedge m'\_dest == dest(m')$ will be added, and then a $pch\_vio_u!$-labeled TA edge that leads to l_0 will be added in $A_{u,i}$. See Fig. 18(b), TA location `Rst1` and edges $(l_1, \texttt{Rst1})$, $(\texttt{Rst1}, l_0)$.

In the above rule, the auxiliary binary channel pch_vio_u (meaning "prechart violation" of chart L_u) is used to notify the coordinator automaton $Coord_u$ of the cold violation. The $reset_u!$-labeled broadcast edge will be added in $Coord_u$. See Fig. 18(c), TA edges $(l_0, \texttt{Rst}), (\texttt{Rst}, l_0)$. In the prechart of L_u, for all positions s on all instance lines I_t such that $Pch\_top_{L_u,I_t} + 1 \leq s \leq Pch\_bot_{L_u,I_t}$, we add a $reset_u$?-labeled edge from l_s to l_0 in $A_{u,t}$. See Fig. 18(a), TA edges (l_1, l_0), (l_3, l_0).

If a message violates the event partial order in the main chart, then it is a hot violation. Once this happens, the corresponding TA will immediately arrive at a deadend error location (`Err`) (Table 2, row 4, lower part).

R7 If at position k on the main chart portion of instance line I_i of chart L_u, there is a receiving of an m-labeled message from instance I_j, then for all m'-labeled messages in L_u such that $m' \neq m$, an m'?-labeled outgoing TA edge which arrives at a deadend error location will be added to location l_{k-1}. See Fig. 18(b), locations `Err1`, `Err2` and edges $(l_4, \texttt{Err1})$, $(l_4, \texttt{Err2})$.

(5) Prechart pre-matching

According to the semantics for invariant mode LSC chart, minimal events in the prechart are constantly being matched for. For example in Fig. 4, $m_1 \cdot m_1 \cdot m_4 \cdot m_2$ is a matching sequence for the second incarnation of this chart under the invariant mode.

R8 If at position 1 on the prechart portion of instance line I_i of chart L_u, there is a sending of an m-labeled message to instance line I_j at its position 1, then an m!-labeled self-loop edge that carries assignment "$m\_src := I_i, m\_dest := I_j, prematch_u$ $:=$ `true`" will be added to location l_0 of $A_{u,i}$. If location l_0 in $A_{u,i}$ has an invariant, then it will be enhanced with a further constraint "$prematch_u ==$ `false`". See Fig. 18(a), location l_0.

Similarly, if there is a receiving of an m-labeled message from I_j, then we add to l_0 an m?-labeled edge, followed by an intermediate committed location which has invariant "$(m\_src == I_j) \wedge (m\_dest == I_i) \wedge (prematch_u ==$ `true`$)$", and then an internal transition edge with assignment "$prematch_u :=$ `false`" leading back to l_0 (Fig. 18(b)).

The flag boolean variable $prematch_u$ is initialized to `false`. Once it is set `true`, it means that chart L_u is currently undergoing a process of prechart pre-matching.

For simplicity, the semantics of prechart pre-matching has not been considered in Section 3.2.1. A remedy to this is to add one more bullet to the "silent step" case, stating that an m-consuming advancement step will just remain at the top cut $\top$.

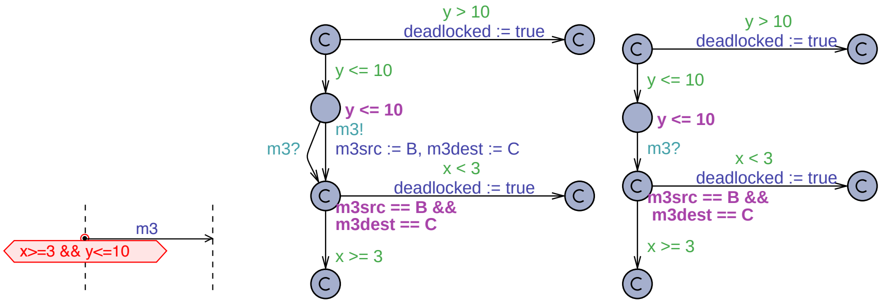
5.1.3 Dealing with time

For time-enriched LSCs, there are further constructs (i.e., clock constraints and clock resets) to be considered during the translation. To mimic the behaviors of each clock constraint and clock reset in an LSC chart, we use a linked sequence of edges in the corresponding timed automaton. The atomicity of executing this sequence is ensured by the UPPAAL feature of committed location.

(1) Translation of guards (clock constraints)

If an instance line has an m-labeled message sending that is guarded by a clock constraint, then a natural idea is to put this constraint on the m!-labeled edge of the translated TA. While this is feasible in the “one-TA-per-chart” translation method, it does not work in the “one-TA-per-instance line” method of this section. The reason is that we need to use a broadcast channel m to handle the inter-chart coordination (see Section 5.1.2); however, due to the restriction of UPPAAL, broadcast channels cannot carry clock constraints [6]. To overcome this problem, in the translated TA, the upper bound constraint (if any) such as $x \leq 5$ will be tested prior to the message sending, and the lower bound and/or clock difference constraints (if any) such as $x \geq 3$ and $x - y \leq 2$ will be tested immediately after the message sending (Table 2, row 5).

R9 If at position k on the main chart portion of instance line I_i of chart L_u there is a sending of an m-labeled message which is guarded by a clock constraint (Fig. 19(a)), then in $A_{u,i}$ there will be first an intermediate committed location for upper bound constraint test. If true, then the next will be a normal location l_k with the upper bound constraint as the location invariant, which will in turn be immediately followed by a message sending edge. Finally, there will be another intermediate committed location for lower bound and/or clock difference constraint test. See Fig. 19(b).



(a) A guarded message (b) TA for the sending instance line (c) TA for the receiving instance line

Fig. 19 Translating a guarded message to TA fragments

For the receiving position of the guarded message, the translation is similar, see Fig. 19(c).

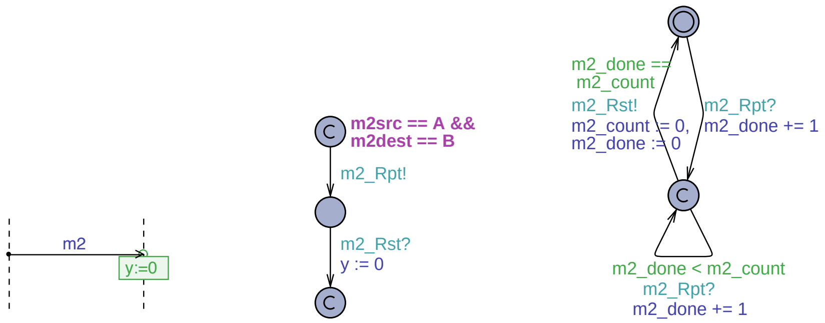
A clock constraint in the prechart portion is translated slightly differently. Since in the prechart the messages are being monitored rather than being enforced, there will be no pre-transition upper bound constraint test. Instead, it will also immediately follow the message sending transition and will be merged with the lower bound and/or clock difference constraint test.

(2) Translation of assignments (clock resets)

In a time-enriched LSC chart, an assignment (i.e., a set of clock resets) should take place immediately after the synchronization of the message occurrence that it is attached to. But in the translated TA, it cannot be put on the very edge that corresponds to the message sending/receiving, because clock resets should not occur before the lower bound or clock difference constraint test which is supposed to happen *immediately after* the message synchronization. Neither can we append the TA edge that carries the assignment to the destination locations of the lower bound or clock difference constraint test, because if several identically-labeled message occurrences are simultaneously enabled in their respective charts where those charts have different guards and/or assignments for those message occurrences (Fig. 5, the m_3-labeled message occurrences), then there could be race conditions (e.g., the assignment $x := 0$ that is attached to m_3 in Fig. 5(a) could happen *before* the lower bound test $x \geq 3$ in Fig. 5(b)). Clearly, this is not what we want.

To model the clock resets properly, for each message $m \in \Pi$ in an LSC system, if m is ever associated with an assignment, then we use a dedicated process (TA) A_m to coordinate all the clock resets of the corresponding message occurrences that are engaged in the same broadcast synchronization on m. When the broadcast synchronization happens, we use an integer variable *m_count* to bookkeep how many instance lines in the LSC system have participated in this broadcast synchronization. Whenever one of these instance lines is done with its lower bound and/or clock difference constraint test (if any), it will immediately notify A_m of its completion using a binary channel *m_Rpt* (i.e., “reporting to A_m”), and after that it will wait there for a synchronization on the broadcast channel *m_Rst* (i.e., “resetting clocks” command from A_m), along with which it can carry out its clock resets. In A_m, an integer variable *m_done* is increased by 1 each time when A_m is notified by an instance line (via *m_Rpt*?). Once *m_done* rises up to *m_count*, A_m will immediately initiate the broadcast synchronization (via *m_Rst*!) (Table 2, row 6).

R10 If at position k on instance line I_i of chart L_u there is a receiving of an m-labeled message which has clock resets (Fig. 20(a), instance line on the right), then there will be first an *m_Rpt*!-labeled outgoing edge from location l_k in $A_{u,i}$, then a normal location, and then an *m_Rst*?-labeled outgoing TA edge that carries the clock resets. See Fig. 20(b).



(a) A message with clock reset

(b) TA for the receiver

(c) The dedicated TA A_{m_2}

Fig. 20 Translating a message with clock reset to TAs

The dedicated TA A_m just waits for all the relevant instance lines to be done with their lower bound and/or clock difference constraint tests, and then synchronizes them for clock resets (Fig. 20(c)).

(3) Just-in-Time message upper bound constraint test

When time-enriched LSC charts have upper-bound clock constraints, there are conditional tests before message sending/receiving in the translated timed automata. Given an m-labeled message occurrence in a chart, a potential problem is that in the translated timed automata for the sending and receiving instance lines, the TA locations that correspond to the sending and the receiving positions of this message may not be ready for this message synchronization at the same time (Fig. 5(b), the m_1-labeled message occurrence). In the symbolic exploration of the state space of the translated network of timed automata, problems will arise if the upper bound of some message sending/receiving is tested when actually it *should not* at that time. For example in Fig. 5, assume that a message sequence $m_1 \cdot m_2$ has been observed, and both charts have just entered their main charts, respectively. Note that a next m_1 is not enabled at the current cut (w.r.t. $\preccurlyeq$). But according to our translation method mentioned earlier in this section (5.1.3), its guard will incorrectly add a further constraint "$x \leq 2$" to "$(x \leq 5) \wedge (x \geq 3 \wedge y \leq 10)$". Consequently, in this example all possible paths will end up with hot violations.

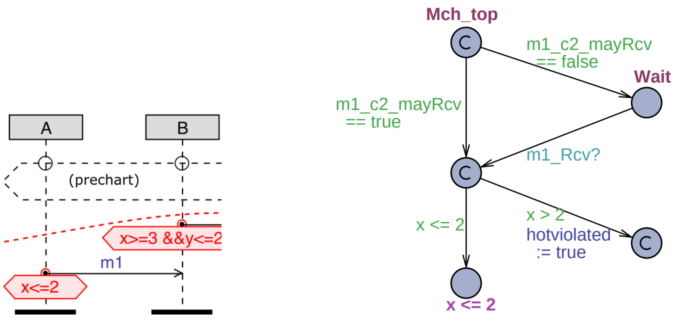
To avoid this kind of premature tests of upper bound constraints for message occurrences, we associate each message occurrence *mo* in each chart L_u with two flag boolean variables *mo_u_maySnd* and *mo_u_mayRcv*, denoting whether this message may be sent or received in chart L_u, respectively. The upper bound constraint of *mo* can be tested only if both flag variables evaluate to `true`.

R11 If at position k on instance line I_i of chart L_u there is a sending of message occurrence *mo* which has a clock constraint (Fig. 19(a)), then there will be a preceding edge carrying the predicate "*mo_u_mayRcv* $==$ `true`". Once this message synchronization is fired, *mo_u_maySnd* will be cleared.

For the receiving instance line of message occurrence *mo*, the corresponding predicate will be "*mo_u_maySnd* $==$ `true`".

If *mo* is a minimal event in the prechart (resp. main chart), then *mo_u_maySnd* and *mo_u_mayRcv* will have initial values `true` (resp. will be set to `true` by the $activate_u$ synchronization). If *mo* is not a minimal event, then the flag variables will be set to `true` by its predecessor events.

Given a message occurrence *mo*, if the predecessor positions of the head/tail positions of *mo* are also the head/tail (or tail/head) positions of another message occurrence, or *mo* is a minimal event, then *mo_u_maySnd* and *mo_u_mayRcv* will be both `true` prior to the constraint tests. Otherwise, their truth values may differ, e.g. in Fig. 21(a), message occurrence m_1 for the current cut (the dashed free line). In this case, the translated TA $A_{2,A}$ will go to location `Wait` to "sleep", and will then be woken up by a dedicated message mo_1*_Rcv* that is sent by the TA $A_{2,B}$ (Fig. 21(b)).



(a) An LSC fragment

(b) TA fragment for instance line *A*

Fig. 21 An LSC fragment of Fig. 5(b) and the corresponding TA fragment for its instance line *A*

5.1.4 Translating non-message simregions

In Sections 5.1.2 and 5.1.3, the simregions take the forms of pure message occurrences and message occurrences that are associated with conditions and/or assignments, respectively. In this section, we consider the non-message simregions. As mentioned in Section 3.1, we adopt the ASAP semantics for these simregions. Depending on how many instance lines a non-message simregion is anchored to, we translate it in different ways:

- one instance line. A single edge will be created in the timed automaton that corresponds to this instance line to carry the condition and/or assignment of the non-message simregion;
- two instance lines. We treat the non-message simregion as if there were an implicit message that carries the condition and/or assignment, and this message were sent from one to the other instance line. Accordingly, we create a fresh auxiliary binary channel in the timed automata, and translate similarly as in Section 5.1.3;
- three or more instance lines. We treat the non-message simregion as if there were an implicit synchronization among them that carries the condition and/or assignment. Accordingly, we create a fresh auxiliary broadcast channel in the timed automata, and translate similarly as in Section 5.1.3.

5.1.5 Translating monitored charts

In comparison with a driving universal chart, the translation of a monitored chart is different in the point that a monitored chart only “listens to” the messages in the LSC system and never emits messages by itself. When translating such a chart to a network of timed automata, if at position s of instance line I_k there is a sending of an m-labeled message, then instead of adding an m!-labeled TA edge from l_{s-1} to l_s, we only add an m?-labeled one in between.

5.2 Complexity of translated timed automata

Let LS be a set of LSC charts $L_1, L_2, \ldots, L_n$, and let NTA_{LS} be the translated network of timed automata. Let $inst(L_i)$, $ML(L_i)$, $MA(L_i)$ and $MO(L_i)$ denote the set of instance lines, the set of message labels (i.e., "signals"), the message alphabet and the set of message occurrences of chart L_i, respectively.

Table 3 summarizes the complexity of the outcomes (more precisely the worst-case outcomes) of the translation in different settings, namely, a single LSC chart or an LSC system; untimed LSC chart or time-enriched LSC chart.

Table 3 The complexity of the outcomes of LSC-to-TA translation ("one-TA-per-instance line")

number of	A single chart L	
	untimed chart	time-enriched chart
TAs	$\lvert inst(L)\rvert + 1$	$\lvert inst(L)\rvert + \lvert MA(L)\rvert + 1$
channels	$\lvert ML(L)\rvert + 2 \cdot \lvert inst(L)\rvert + 4$	$\lvert ML(L)\rvert + 2 \cdot \lvert inst(L)\rvert + 4 + 3 \cdot \lvert MA(L)\rvert$
auxiliary variables	$2 \cdot \lvert MA(L)\rvert + 2$	$4 \cdot \lvert MA(L)\rvert + 2 \cdot \lvert MO(L)\rvert + 2$
number of	A set of driving charts $L_1, \ldots, L_n$	
	untimed charts	time-enriched charts
TAs	$\sum_{i=1}^{n}(\lvert inst(L_i)\rvert + 1)$	$\sum_{i=1}^{n}(\lvert inst(L_i)\rvert + 1) + \lvert \bigcup_{i=1}^{n} MA(L_i)\rvert$
channels	$\lvert \bigcup_{i=1}^{n} ML(L_i)\rvert +$ $\sum_{i=1}^{n}(2 \cdot \lvert inst(L_i)\rvert + 4)$	$\lvert \bigcup_{i=1}^{n} ML(L_i)\rvert + \sum_{i=1}^{n}(2 \cdot \lvert inst(L_i)\rvert + 4) +$ $3 \cdot \lvert \bigcup_{i=1}^{n} MA(L_i)\rvert$
auxiliary variables	$2 \cdot \lvert \bigcup_{i=1}^{n} MA(L_i)\rvert + 2n$	$4 \cdot \lvert \bigcup_{i=1}^{n} MA(L_i)\rvert + 2 \cdot \sum_{i=1}^{n} \lvert MO(L_i)\rvert + 2n$

For a time-enriched LSC system, we analyze the complexity of the translated network of timed automata as follows:

Let the set LS of time-enriched charts $L_1, L_2, \ldots, L_n$ have messages $m_1, m_2, \ldots, m_k$, message occurrences $mo_1, mo_2, \ldots, mo_s$, and instance lines $I_{i,1}, I_{i,2}, \ldots, I_{i,in_i}$, where $1 \leq i \leq n$, $in_i = \#(inst(L_i)) = |inst(L_i)|$.

According to Section 5.1.2 ("Handling intra-chart coordinations") and Section 5.1.3 ("Translation of assignments"), in the worst case (i.e., each message has been associated with some clock resets somewhere in LS), the translated network of timed automata will be $NTA_{LS} = \{A_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq \#(inst(L_i))\} \cup \{Coord_i \mid 1 \leq i \leq n\}$ $\cup \{A_{m_i} \mid 1 \leq i \leq k\}$. Therefore, the number of timed automata is $\sum_{i=1}^{n}(|inst(L_i)|) + n + |\bigcup_{i=1}^{n} MA(L_i)|$ (Table 3, lower part right column).

According to rule R2, each message label corresponds to a channel in NTA_{LS}. According to R3, R4 and R6, there will be a set of auxiliary channels $Aux = \{pch\_over_{u,i}, mch\_over_{u,i} \mid 1 \leq u \leq n, 1 \leq i \leq \#(inst(L_u))\} \cup \{activate_u, over_u, pch\_vio_u, reset_u \mid 1 \leq u \leq n\}$ that will be used in NTA_{LS}. According to Section 5.1.3 ("Translation of assignments"), in the worst case, there will be a set of auxiliary channels $Aux' = \{m_i\_Rpt, m_i\_Rst, m_i\_Rcv \mid 1 \leq i \leq k\}$ for translating clock resets. Therefore, in the worst case, the number of channels in NTA_{LS} will be $|\bigcup_{i=1}^{n} ML(L_i)| + \sum_{i=1}^{n}(2 \cdot |inst(L_i)| + 4) + 3 \cdot |\bigcup_{i=1}^{n} MA(L_i)|$.

According to Section 5.1.2 ("Basic mapping rules"), there will be a set of auxiliary variables $\{m_i\_src, m_i\_dest \mid 1 \leq i \leq k\}$. According to rules R3 and R8, there will be a set of auxiliary variables $\{prematch_u, dInst_u \mid 1 \leq u \leq n\}$. According to Section 5.1.3 ("Translation of assignments"), in the worst case, there will be auxiliary variables $\{m_i\_count, m_i\_done \mid 1 \leq i \leq k\}$. Furthermore, according to R11, there will be auxiliary variables $\{mo_i\_maySnd,\ mo_i\_mayRcv \mid 1 \leq i \leq s\}$. Therefore, the total number of auxiliary variables in NTA_{LS} may be up to $(2 \cdot |\bigcup_{i=1}^{n} MA(L_i)|) + 2n +$ $(2 \cdot |\bigcup_{i=1}^{n} MA(L_i)|) + (2 \cdot \sum_{i=1}^{n} |MO(L_i)|)$. □

The complexities for the other three settings can be analyzed similarly.

Remark 1 Although during the translation we introduce some auxiliary (coordinator) automata, auxiliary channels and auxiliary integer data variables, we do not introduce auxiliary clock variables. This is important because the time complexity of model checking timed automata is much more sensitive to the number of clocks and the clock upper bounds thereof in the system (e.g., reachability and emptiness problems have a factorial growth with the number of clocks). Furthermore, among all the auxiliary variables, $(n + 2 \cdot \sum_{i=1}^{n} |MO(L_i)|)$ of them are boolean variables, and the rest of them are bounded integer variables that have relatively small ranges. Still further, rather than being able to take arbitrary values in their respective ranges, these auxiliary integer variables are inter-correlated and thus have a limited number of value combinations. All these indicate that the translated network of timed automata does not have an overwhelmingly large state space as it first appears to have.

Remark 2 In Sections 3 – 5, we assume a liberal object interaction context in the sense that a message label (or "signal") can be viewed as a member function of a class in object-oriented programming; and instances of other classes can call the method of a certain instance of the method-owner class (i.e., to send a message to that instance). Therefore in the translation, each message m needs to be associated with two bounded integer variable *m_src* and *m_dest*, representing the particular sending and receiving instances, respectively.

In case that object-orientation is not a major concern, we can make the "message sender/receiver uniqueness" assumption, i.e., each message label corresponds to a unique sending process and a unique receiving process in the communicating system. In other words, different messages must carry different message labels. Under this assumption, for a set of untimed (resp. time-enriched) charts $L_1, L_2, \ldots, L_n$, in Table 3 the number of needed auxiliary bounded integer variables will decrease to $2n$ (resp. to $2 \cdot |\bigcup_{i=1}^{n} MA(L_i)| + 2 \cdot \sum_{i=1}^{n} |MO(L_i)| + 2n$).

Remark 3 In Section 5.1.2 ("Basic mapping rules"), for each message $m \in \Pi$ we assign two bounded integer variables *m_src* and *m_dest*. Although this handling has good readability and is easily comprehensible, it is not really necessary to assign so many auxiliary integer variables even if we do not make the "message sender/receiver uniqueness" assumption.

On one hand, if a message $m \in \Pi$ is always sent from one instance line to another, then we do not need to use *m_src* and *m_dest* at all. On the other hand, if a message $m \in \Pi$ has different sending and/or receiving instance lines, then let the number of sender/receiver combinations be k. It suffices to associate m with only one auxiliary bounded integer variable *m_sd* ("src-dest" of m) rather than *m_src* and *m_dest*. The range of *m_sd* could be $[0, k-1]$. The constraints and updates on *m_src* and *m_dest* in Section 5 can be modified accordingly. In this way, the state space of the translated network of timed automata will be further reduced.

Remark 4 Section 4.2 shows that the observer timed automaton which is obtained by our first ("one-TA-per-chart") translation has exponential space complexity. Since this is due to the inherent complexity of the LSC formalism, the translation outcome cannot be simpler. In Section 5.1, the translated network of timed automata also has exponential space complexity when they are composed in parallel. A justification is that each cut of the chart corresponds to a location vector of the network of timed automata (this is shown in the proofs of Theorem 5 and its lemmas).

Since a number of auxiliary automata, locations and variables have been introduced, the outcome of our "one-TA-per-instance line" translation may have a larger state space than the first approach. However, the advantage of this approach is also clear. We do not need to explicitly generate and store a (possibly huge) global transition system (i.e., the parallel composed timed automata), whose state space does not necessarily need to be fully explored by the on-the-fly verification algorithms of UPPAAL.

5.3 Equivalence of LSC and TAs

Theorem 5 *Let LS be a set of time-enriched LSC charts whose message alphabet is Π, and let NTA_{LS} be the translated network of timed automata which have a set Act of normal and auxiliary channels. Then* $\forall \gamma_1 \in (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{\omega} . ((\gamma_1 \models LS) \Rightarrow \exists \gamma_2 \in (Act \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{\omega} . (\gamma_2 \models NTA_{LS}) \wedge (\gamma_2|_{(\Pi \cup \mathbb{R}_{\geq 0})} = \gamma_1|_{(\Pi \cup \mathbb{R}_{\geq 0})}))$, *and* $\forall \gamma_2 \in (Act \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{\omega} . ((\gamma_2 \models NTA_{LS}) \Rightarrow \exists ! \gamma_1 \in (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{\omega} . (\gamma_1 \models LS) \wedge (\gamma_2|_{(\Pi \cup \mathbb{R}_{\geq 0})} = \gamma_1|_{(\Pi \cup \mathbb{R}_{\geq 0})}))$.

Theorem 5 indicates that each accepted timed trace γ_1 in *LS* uniquely corresponds to a cluster of accepted timed traces in NTA_{LS}. All these traces correspond to exactly the same restricted trace on the message alphabet and time delays $(\Pi \cup \mathbb{R}_{\geq 0})$ as γ_1 does.

The lemmas for theorems in Sections 5.3 and 5.4, and the proofs of these lemmas and theorems can be found in Appendix D.

5.4 Verification problems

In the context of scenario-based verification, we would like to ask whether a system that is modeled as a set *LS* of driving universal charts satisfies the requirements that are specified as a separate monitored universal or existential chart L'. Here L' will be translated to a network of observer timed automata $NTA_{L'}$, i.e., they only "listen to" the messages in NTA_{LS}, and never emit messages to NTA_{LS} by themselves. We let $Coord_{L'}.Mch\_top$ and $Coord_{L'}.Mch\_bot$ denote that the coordinator automaton $Coord_{L'}$ for chart L' is in its locations *Mch_top* and *Mch_bot*, respectively.

We have the following two main theorems:

Theorem 6 *Let LS be an LSC system, and L' be a monitored universal chart, then* $LS \models L' \;\Leftrightarrow\; (NTA_{LS}\,||\,NTA_{L'}) \models (Coord_{L'}.Mch\_top \leadsto Coord_{L'}.Mch\_bot)$.

Theorem 6 indicates that in order to check whether an LSC system satisfies the requirements in a separate universal chart, we only need to check whether the responsiveness CTL property is satisfied by the parallel composition of their corresponding networks of timed automata.

Theorem 7 *Let LS be an LSC system, and L' be a monitored existential chart, then* $LS \models L' \;\Leftrightarrow\; (NTA_{LS}\,||\,NTA_{L'}) \models \mathsf{E}\Diamond\; Coord_{L'}.Mch\_bot.$

Theorem 7 indicates that in order to check whether an LSC system satisfies the requirements in a separate existential chart, we only need to check whether the parallel composition of their corresponding networks of timed automata has a trace that can be observed by the existential chart as a satisfying run.

6 Tools and experiments

6.1 "One-TA-per-chart" approach

Based on previous work [33], the approach in Section 4 has been implemented as a prototype LSC editor and LSC-to-TA translator, which have been integrated into a recent version of the UPPAAL frontend client (GUI) and backend verification server [5], respectively. The LSC editor and translator support the LSC elements of instance line, location, message, condition, assignment, simregion, prechart and main chart. Furthermore, the LSC editor supports the creation and instantiation of LSC templates, where the template parameters can be used to parameterize the instance lines and messages in the charts. Both the translation and the composition algorithms have been integrated into the UPPAAL verification server (Fig. 1, left part). In this way, the user achieves scenario-based push-button verification.

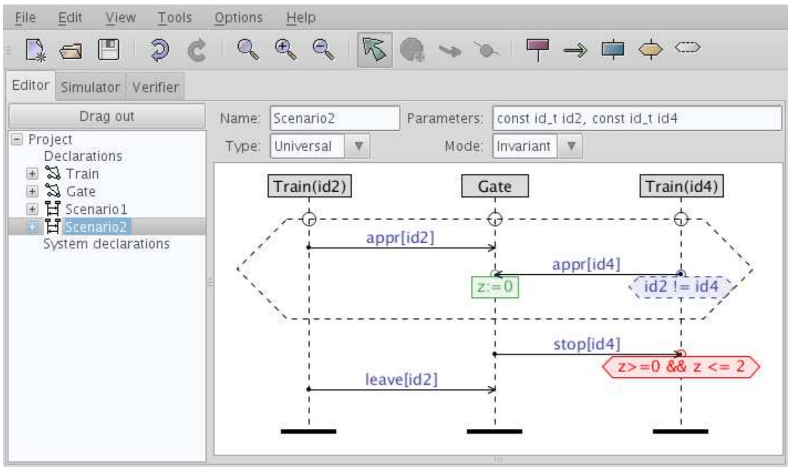
We carry out experiments on the Train-Gate example [40]. The system model consists of a number of Trains, each of which may want to cross the only bridge (i.e., to access a critical resource), and a Gate controller, which ensures correct signalling with the trains (i.e., to sense the approaching and leaving of trains, and to command the trains to stop and to resume). Each train uses a clock variable to keep track of its timings of approaching/leaving the gate and being stopped/resumed by the gate. All these trains are subject to certain timing constraints such that collisions are avoided and a safe separation distance between any two trains is always maintained.

We can use time-enriched LSC charts to capture the scenario-based requirements such as:

- L_1 ("liveness"): Once a train approaches the gate, it must eventually (cross and then) leave the gate; and
- L_2 ("freedom from collisions"): If a second train approaches the gate before the first approaching train completes its crossing, then the gate must signal the second train to stop within a certain period of time, and after that the first train must leave the gate (Fig. 22, LSC template "Scenario2").

The original system TA models, the LSC requirements, and the translated and composed system models can be found in [30].

Note that the LSC charts can have their own (i.e., “private”) clocks that do not appear in the original system models, e.g., clock z in the chart template “Scenario2” of Fig. 22.



Name: Scenario2
Parameters: const id_t id2, const id_t id4
Type: Universal
Mode: Invariant

Fig. 22 A scenario-based “freedom from collisions” requirement in the UPPAAL-integrated LSC editor

Table 4 presents the experimental results of verifying the Train-Gate system against the scenario-based requirements L_1 and L_2. The time overheads and memory consumptions as listed in the four rightmost columns of Table 4 are the sums for the three consecutive phases, i.e., LSC-to-observer timed automaton translation, system and observer timed automaton instrumentation and composition, and model checking the composed system against the automatically extracted CTL formula. In comparison, columns 2 to 5 of Table 4 present the time and memory consumptions for “pure verification”, i.e., merely to model check the translated and composed system (which can be saved as a network of UPPAAL TAs (.xml model file) by the translator) against the automatically extracted CTL formula (which can also be saved as a UPPAAL query (.q property file) by the translator). Furthermore, we vary the number of trains in the system to see how this approach scales. Since each train has a clock variable and the monitored LSC chart may have a fresh clock variable, the total number of clocks in the system is the number of the trains plus one.

As can be seen in Table 4, model checking an LSC-specified requirement against a TA-modeled real-time system of up to 10 clocks (i.e., 9 trains plus 1 LSC chart) can be achieved on an ordinary PC. Considering that the models of many real-life applications have only a few clocks, our approach can be of practical value. Furthermore, it appears that both the time overheads and the memory consumptions increase exponentially with the size of the system and number of clocks. Specifically, when the number of trains increases to 10, UPPAAL does not issue a pass/fail verdict after it has been running for more than 12 hours and consumed 2.2GB memory. Further-

Table 4 Experimental results of scenario-based verification of the Train-Gate example

# of trains	performance results ⟨CPU time: sec, memory: KB⟩							
	conventional approach in Uppaal (“pure verification”)				approach of Section 4 (“translation + composition + verification”)			
	L_1		L_2		L_1		L_2	
2	< 0.01	4108	< 0.01	4232	< 0.01	4220	< 0.01	4504
3	0.01	4208	0.01	4460	0.02	4492	0.01	4672
4	0.03	4324	0.03	4652	0.04	4936	0.03	5240
5	0.23	4664	0.23	4936	0.24	5058	0.24	6080
6	0.68	7252	0.73	7640	0.69	8660	0.75	8908
7	4.38	29892	4.46	27604	4.39	31884	4.48	29764
8	41.66	232924	39.49	191764	41.83	235504	39.61	194428
9	508.14	2127884	404.07	1700360	508.38	2167420	404.24	1703848
10	(> 12 hours, ≈ 2.2GB)							

Experiment platform: Intel Core 2 Duo P8700 CPU (2×2.53GHz), 4 GB RAM; Ubuntu 10.4, Uppaal 4.1.3.

more, system monitoring shows that the memory usage is almost stabilized within this long procedure (12+ hours). All these seem to indicate that time complexity is more of a problem in this approach.

By comparing the results of the “pure verification” and our scenario-based verification, it is clear that the latter approach is only a little more expensive than the former one (Table 4). This is reasonable because translation and composition as syntactical level manipulations are far less computation-intensive than the subsequent model checking. Considering that manual construction of observer timed automata is in general a time-consuming and error-prone process, our approach achieves *automated* scenario-based verification at the expenses of only a little extra efforts.

As a classic benchmark, the Train-Gate example has been widely examined in real-time system modeling and verification, including LSC-based verification. The scenario-based verification problems in [34] are very similar to ours, though the authors use a variant of the Train-Gate TA models and define LSC in a different manner (e.g., their LSC has subcharts and conditional constructs, but no guarded messages with clock resets as in this article). For a Train-Gate system of 4 trains and an LSC scenario that is very similar to our “freedom from collisions” property L_2, their verification takes 6.4 seconds (dual Pentium 4 Hyper-Threading 2.8 GHz CPUs). As can be seen in Table 4, we need 0.03 second. Although it is not fair to compare them directly, the results may be indicative of the efficiency of our approach.

6.2 “One-TA-per-instance line” approach and comparison with previous approach

To realize the approach in Section 5, we have built a GUI-based LSC editor, with which we can construct either universal or existential charts, and we have implemented a prototype command line LSC-to-TA translator (“one-TA-per-instance line” approach), which is capable of batch translation of the prescribed LSC charts. The translator-generated timed automata and extracted CTL formulas comply with the

Uppaal timed automaton and query language syntax, and can thus be fed into Uppaal. The LSC editor, the LSC-to-TA translator and the Uppaal model checker collectively constitute a tool chain.

We still use the Train-Gate problem to illustrate how our "one-TA-per-instance line" approach performs. The system consists of two trains and one gate. The scenario-based interaction behaviors are modeled as five driving LSC charts. The scenario-based requirement is captured in a separate monitored LSC chart. We let this requirement be freedom from collisions (Section 6.1, property "L_2").

The set LS of driving charts and the monitored chart L' are translated to networks of timed automata NTA_{LS} and $NTA_{L'}$ (Table 5, lower part), respectively. The models of the driving/monitored LSC charts and the translated TAs can be found in [30].

Table 5 Experimental results of the approaches of Sections 4 and 5 using the same Train-Gate example

	original and LSC-translated TA models						model checking	
	TAs	locations	edges	channels	clocks	variables	time (s)	memory (KB)
Train-Gate system (original TA models)	*3*	*13*	*17*	*8*	*2*	*4*	0.007	4232
monitored chart "L_2" (approach of Sect. 4)	1	5	32	6	1	4		
5 driving charts (approach of Sect. 5)	23	251	352	65	5	57	3.93	25848
monitored chart "L_2" (approach of Sect. 5)	5	55	75	16	1	19		

Experiment platform: Intel Core 2 Duo P8700 CPU (2×2.53GHz), 4 GB RAM; Ubuntu 10.4, Uppaal 4.1.3.

We also carry out comparative study of the "one-TA-per-chart" and "one-TA-per-instance line" approaches using the same configuration of the Train-Gate problem as above (i.e., two trains and one gate). We compare the two approaches in terms of the sizes of the translated timed automata and the performances of the model checking that are subsequently performed on these timed automata (Table 5).

As we can see, the amounts of translation outcomes of the driving charts (Table 5, middle columns data, row 3) and of the monitored chart (row 4) using the "one-TA-per-instance line" approach are roughly in proportion to the numbers of the charts (5 and 1 in this example, respectively). The reason is that these two kinds of charts are translated in basically the same way (Section 5.1.5). However, the outcome of the "one-TA-per-instance line" translation (Table 5, middle columns data, row 4) is more complex than that of the "one-TA-per-chart" translation (row 2). The reason is that many auxiliary channels and variables are needed to properly handle intra-/inter-chart coordinations. In other words, the benefit of "distributed" translation usually comes along with this kind of structure complication of translation.

Also from Table 5 we notice that model checking the translated five driving charts against the translated monitored chart using the "one-TA-per-instance line" approach (Table 5, right columns data, row 2) requires more CPU time and memory than model checking the original Train-Gate TA system against the translated monitored chart using the "one-TA-per-chart" approach (row 1). This is mainly because that the translated network timed automata of the five driving charts are far more complex than the original TA models of the Train-Gate system [40], which consists of only three simple TAs that have 2 clocks, 8 channels and 4 variables (Table 5, middle columns data, row 1).

At a first glance it seems that our "one-TA-per-instance line" approach is very likely to suffer from scalability problems: even for a small system, we need to create a relatively large number of LSC charts to model the system behaviors and to specify the user requirements. In this way, the translation will yield a large network of timed automata. Consequently, we are left in doubt whether the efforts of dealing with this TA system will outweigh the benefits of using LSCs. However, a close examination of the LSC charts in this Train-Gate example reveals that we need to create so many charts (and the respective clock variables for these charts) mainly because that our time-enriched LSC in its current form has only a limited number of language constructs. Consequently, a single LSC chart captures only a small piece of system behaviors or user requirements. If we extend it with e.g. the control structures such as branching and looping, symbolic mechanisms such as symbolic messages and symbolic instances, and with forbidden and ignored messages, then our LSC models themselves will be much more succinct than what they are now (i.e., fewer charts and fewer clocks will be needed to model the same system and specify the same requirement). The numbers of translated timed automata and the clock variables can hopefully be kept within reasonable sizes. This likelihood and tendency has been shown by previous work on the "one-TA-per-chart" translation of the richer LSC models [33].

7 Translating iterative mode charts

When translating LSC charts to timed automata in Sections 4.1 and 5.1, we considered only the invariant activation mode.

As mentioned in Section 3.1, a universal chart under the iterative mode requires that, as long as the main chart is currently active, the prechart will not be monitored for further satisfaction. Compared with the invariant mode, satisfaction of a universal chart under the iterative mode is a weaker requirement. In other words, given an LSC chart and a message sequence, it may happen that under the invariant mode, the chart hot-violates this sequence, but under the iterative mode, the chart does not. For example, for the LSC chart in Fig. 23(a), the message sequence "$m_1 \cdot m_2 \cdot m_1 \cdot\ m_3$" is hot-violated by the second incarnation of this chart under the invariant mode. However, it is not violated by any incarnation under the iterative mode, because under this mode the second incarnation is "killed" immediately after the prechart of the first incarnation is matched by the sub-sequence "$m_1 \cdot m_2$".

To conduct the "one-TA-per-chart" translation for *iterative* mode charts, we convert a chart into a timed automaton similarly as in Section 4.1 (except that we discard the "prechart pre-matching" step). Now we maintain multiple copies of the translated TA. Each of these copies will become a chart incarnation (i.e., a "live" copy) when necessary. When a new message arrives, the existing chart incarnations (live copies) will react to it as usual, and besides, at most one of the remaining chart copies will be incarnated upon this message (this is ensured by using the auxiliary binary semaphore *MoreIncarnations*, the "allocating" variable *NxtIncarnation*, and the function *IncNxtIncarnation()* that updates *NxtIncarnation*, see Fig. 23 and the UPPAAL declarations below).

The dynamics of the iterative mode activation is implemented by using a fresh auxiliary broadcast channel kill. Once an incarnation of the chart notices that its prechart has been successfully matched, it will immediately initiate a broadcast synchronization on kill to reset (i.e., to "kill") all other live copies that are still progressing in their respective prechart portions.

As an example, if we interpret the LSC chart in Fig. 23(a) under the iterative mode, then Fig. 23(b) is the corresponding TA template `IterativeModeTemplate()` (for better legibility, we have omitted the guards on the sending and receiving instance lines). This template has a template parameter

```
int[0, MaxIncarnationNum - 1] thisChartCopy
```

and it will be instantiated as follows when we make the system (component) declarations in UPPAAL:

```
// There are at most two incarnations.
Chart0 = IterativeModeTemplate(0);
Chart1 = IterativeModeTemplate(1);

system Chart0, Chart1;
```

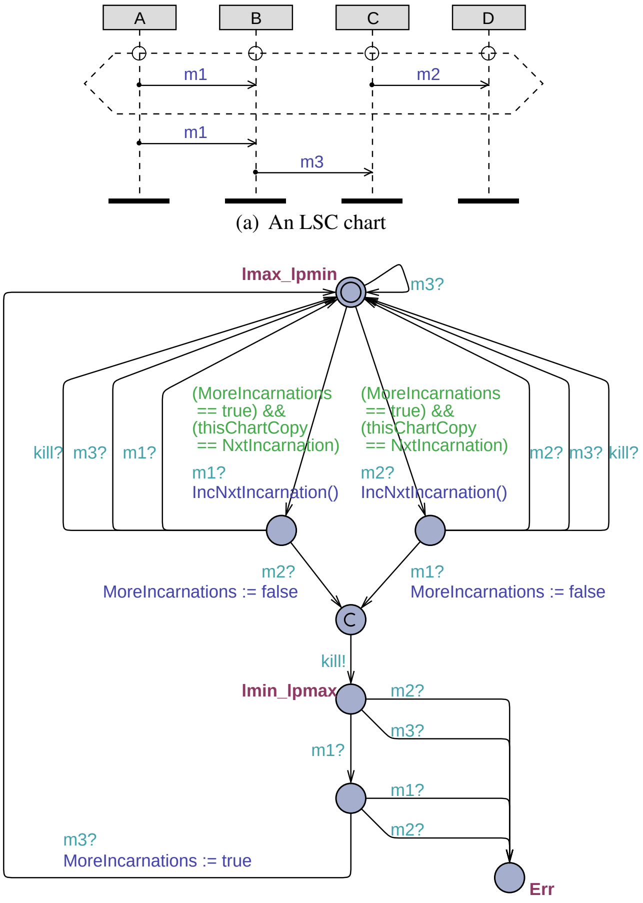
The global (variable, channel and function) declarations of the above system in UPPAAL are as follows:

```
const int MaxIncarnationNum := 2; // The maximal number of incarnations for this chart.

int[0, MaxIncarnationNum - 1] NxtIncarnation := 0; // Which chart copy should be the
                                                    // next incarnation?
bool MoreIncarnations := true; // "Should the Pch be monitored or not?"

broadcast chan m1, m2, m3; // The message labels in the LSC chart
broadcast chan kill; // "kill" is a fresh auxiliary channel.
                     // Once an incarnation of the LSC chart enters its main chart, it
                     // should immediately reset all other incarnations of that chart.

// The function that shifts the pointer to the next chart incarnation
void IncNxtIncarnation() {
    NxtIncarnation := (NxtIncarnation + 1) % MaxIncarnationNum ;
}
```

The LSC elements such as conditions and assignments can be added and translated similarly as in Section 4.1. Likewise, we can prove that the LSC chart L and the translated timed automata $O_{L,1}, O_{L,2}, \ldots, O_{L,n}$ are behavior-equivalent. Here $O_{L,i}$ denotes the timed automaton for the i-th chart copy, and n is the maximal number of chart incarnations of L. We can also compose these observer timed automata with the original system model S (i.e., a network of timed automata), and thus get the final (modified) network of timed automata $(S' \,||\, O'_{L,1} \,||\, O'_{L,2} \,||\, \ldots \,||\, O'_{L,n})$.

To verify whether system S is satisfied by chart L under the iterative mode, we do the following CTL model checking:

$$(S' \,||\, O'_{L,1} \,||\, O'_{L,2} \,||\, \ldots \,||\, O'_{L,n}) \models (l_{i,min} \rightsquigarrow l_{i,max}), \qquad 1 \le i \le n$$


A
B
C
D
m1
m2
m1
m3
lmax_lpmin
m3?
(MoreIncarnations == true) && (thisChartCopy == NxtIncarnation)
m1?
IncNxtIncarnation()
(MoreIncarnations == true) && (thisChartCopy == NxtIncarnation)
m2?
IncNxtIncarnation()
kill?
m3?
m1?
m2?
m3?
kill?
m2?
MoreIncarnations := false
m1?
MoreIncarnations := false
C
kill!
lmin_lpmax
m2?
m3?
m1?
m1?
m2?
m3?
MoreIncarnations := true
Err

(a) An LSC chart

(b) The translated TA template `IterativeModeTemplate()`

Fig. 23 An LSC chart and its translated timed automaton template (under the iterative mode)

where $l_{i,min}$ and $l_{i,max}$ denote the TA locations that correspond to the minimal and maximal cuts of the main chart of the i-th incarnation of chart L, respectively.

To conduct the “one-TA-per-instance line” translation for *iterative* mode charts, the above general idea also applies. We also maintain multiple copies of the translated timed automaton for each instance line in each driving chart. In this case, each chart L_i will have its kill_i-broadcast channel, and the synchronization on this channel will take place immediately after the $\mathsf{activate}_i$-broadcast synchronization in L_i.

Remark 5 The two activation modes, i.e., invariant and iterative modes, can serve different but complimentary purposes. When used as a monitored chart, a chart is more difficult to be satisfied under the invariant mode than under the iterative mode. This indicates that invariant mode is better for capturing safety-related scenario requirements. On the other hand, when used as a driving chart, an iterative mode chart can act as a *periodic* event generator, i.e., before a sequence of messages in the main chart are generated and dispatched to fulfil some commitment, it will not search for an "assume" pattern and make any new commitments.

8 Related work

8.1 Verifying state/transition systems against scenario-based requirements

Model checking by definition and in its earliest forms takes a state/transition system model and a temporal logic formula as the inputs. To model check state/transition-based real-time systems against complex properties or scenario-based requirements, various approaches have been proposed.

One solution is the *observer automata* (a.k.a. *test automata* [1]) approach, i.e., to construct a number of auxiliary automata to capture the complex properties or scenario-based requirements, and then use these automata to "observe" the original system model. This approach requires that the observer be compatible with the original system model, and that the observation be non-intrusive and efficient (i.e., it incurs as little extra communication and computation overheads as possible).

An observer timed automata approach to real-time system verification is suggested in [1]. This approach has been used to model check practically relevant systems such as the B&O power controller [19] and some timed safety instrumented systems [27]. Case studies thereof indicate that the approach is effective. However, the approach also comes with some limitations:

- Manual construction of observer timed automata could be labor-intensive and error-prone, and this is especially the case when the automata grow large;
- To synchronize with the observer timed automata, the original system models may need to be modified and annotated. During this modification process, some new errors might be introduced. Newly introduced timing errors are especially difficult to diagnose; and
- Since an observer timed automaton and the original system usually engage in "loose" channel synchronizations (i.e., no particular sending and receiving process are specified for a synchronization on a certain channel (message label)), they specify process interactions only *liberally*. To capture non-trivial scenario-based requirements, the synchronizations between the observer automaton and the original system should be carefully designed by using e.g. auxiliary variables, semaphores or locking mechanisms. This is an extra burden for the designers.

Compared with the observer automata approach, in our method the observer automaton is constructed *automatically*, and it is guaranteed to observe the original system in a *non-intrusive* way. Furthermore, the automatically created auxiliary variables (including semaphores) will enable our observer automaton to faithfully reflect the LSC requirements where each message has its particular sending and receiving process.

Scenario-based requirements on state/transition-based system models can also be captured by using the assume-guarantee style visual formalisms such as Triggered

MSC [36], Template MSC [15] or the even richer LSC [18], and then transformed into directly verifiable formalisms. In particular LSCs can be translated into timed Büchi automata (TBA) [25], timed automata [33], temporal logics [25,16,9,26,13, 8] or sequences of LSC elements [34], and the verification problem can be converted to a model checking problem on existing tools [25,9], or solved directly [34].

An early attempt of introducing LSC features such as the initial and iterative activation modes and the activation condition (a Boolean expression which characterizes the state of the system model when the scenario should start) into UML Sequence Diagrams is made by Lettrari and Klose [31]. They develop a tool to monitor and test the executable UML models (i.e., implementations of RHAPSODY UML models). In comparison, our work follow the more mature LSC definition [18] to support also the invariant mode and the notion of prechart, which details the activation conditions under which they apply, and we aim at scenario-based verification.

Damm and Klose [12] propose to use LSCs in combination with STDs (Symbolic Timing Diagrams) to specify scenario-based requirements on STATEMATE models, and then carry out model checking. This methodology has been concretized and implemented in [25], where an LSC chart is transformed into a timed Büchi automaton, which is further transformed into a temporal logic formula. Further descriptions of how LSC as a specification language can be used in a UML verification environment for the RHAPSODY tool are presented in [35]. In the work of [25], in order to specify real-time requirements, timers [4,20] and timing annotations (or delayed intervals) [4] are added to the LSC charts. To enable the transformation, each location of the LSC chart is equipped with a discrete (integer) clock. Since timers can only express timing constraints within a *single* chart and within a *single* process, and delayed intervals can only express the minimal and maximal delays between two *consecutive* locations, these restrict the expression of timing constraints across processes and across charts. Our LSC charts use TA-like real-valued clock variables. This flavor of timing constraint agrees well with the original TA system model, and thus enable smooth translation of timing information into the observer TA, as well as seamless embedding of the observer TA into the UPPAAL verification framework.

An LSC to timed automata translation is proposed in [33]. When the LSC chart is used as a monitored chart, this translation is similar to our "one-TA-per-chart" translation in the sense that they are both based on the notion of LSC cut and its advancements. However, when LSC serves as a modeling language, this method is faced with the cut and configuration blow-up problem. In comparison, our "one-TA-per-instance line" translation method does not need to explicitly enumerate and thus create TA locations for a potentially huge number of cuts.

LSCs can also be translated into temporal logic formulas [16,26,13,8]. For the kernel subset of LSC in [26], it has been shown that existential charts can be expressed using the CTL logic, and universal charts can be expressed using ($\mathsf{LTL} \cap \mathsf{CTL}$) [16,26]. Similar results are achieved in [13]. However, these methods do not handle explicit time in the charts. In [9], LSC is applied in hardware verification, where the system models are given in Verilog and the user requirements are specified as LSCs. These LSCs are translated to LTL formulas and then fed into the verification environment FORMALCHECK. Since LSCs are used to specify hardware protocols at the register transfer level, a discrete clock tick construct is introduced to explicitly represent the passage of system time. Compared with [9], we use real-valued clocks to represent various timing constraints.

As mentioned thus far, verification techniques that are based on LSC-to-temporal logic translation in general tend to suffer from scalability problems. Industrial case studies [23] show that the LTL formulas grow large even for LSCs of moderate size, and thus formal verification becomes expensive. To overcome this limitation, Klose and colleagues [24] investigate *efficient* model checking of Kripke structures against LSC requirements. In our method, since our observer automaton is tightly coupled with the original system, a very simple CTL property $\mathsf{A}\Box(l_{min} \Rightarrow \mathsf{A}\Diamond l_{max})$ can be extracted from the observer automaton to capture the LSC requirements. In this way we avoid translating LSCs to complex temporal logic formulas.

Another line of work [34] is to extract properties from LSCs as sequences of LSC elements, and to develop verification algorithms to check whether these sequences are respected by the FSM computation graph of the TA model that is exported from UPPAAL. However, simultaneous regions (simregions) in their LSCs are used only to model broadcast communications, and conditions cannot be a part of simregions. Our notion of simregion uses the "[condition] [message]/[assignment]" pattern, thus enables smooth translation to a TA edge.

8.2 Verifying object interaction-based systems against scenario-based requirements

In this case, the system is modeled as a set of driving universal LSC charts and the requirement is specified as a set of monitored universal or existential charts. Monitored universal charts should not be hot-violated, and monitored existential charts should be matched at least once [18, 7].

In the execution (or play-out) [18] of scenario-based models, the Play-Engine checks whether the monitored charts are respected. This is enhanced in the *smart* play-out mechanism, where planned state space exploration via model checking is added to the Play-Engine to bypass some avoidable hot violation situations that are caused by some "blind" interactions among the system processes. In a case study of a telecommunication application, Combes and colleagues [10] check whether a set of monitored existential charts can be satisfied by a set of driving charts without violating any of them. Their method is based on the play-out and smart play-out mechanisms in the Play-Engine.

In [37], LSCs are encoded as CSP processes. The CSP verification tool FDR is employed to check whether a set of monitored existential charts can be satisfied by a set of driving universal charts. This work considers untimed charts only.

Wang and colleagues [39] employ constraint logic programming (CLP) techniques to enable the symbolic execution of LSC models. Their implementation supports both universal and existential charts, and supports timing constraints.

Playing-out [18, 39, 10] as a methodology of executing scenario-based models does not have in mind verification as its goal. Although it is possible to check whether a number of existential charts can be satisfied during (smart) playing-out, essentially it is still considered as a kind of (guided) execution. Compared with [18, 37, 10], our method allows the requirements to be specified also as universal charts. Furthermore, compared with [18,37,10], our method uses TA-like clock variables and clock constraints, and thus enables finer characterizations of timing requirements.

9 Conclusions and future work

This article proposes two approaches to the verification of state/transition-based and object interaction-based real-time system models against scenario-based user requirements, respectively. We extend a kernel subset of the LSC language with timed automata-like real-valued clock variables and timing constraints, define its semantics, and use the time-enriched LSC charts both for system modeling and for property specification. By means of behavior-equivalent model transformation and non-intrusive event spying, we convert the scenario-based verification problems to CTL real-time model checking problems in UPPAAL. By doing so we conclude that it is feasible:

- to introduce important notions from the TA to the LSC formalisms to facilitate scenario-based characterization of dense real-time systems;
- to employ the original TA constructs as well as the UPPAAL-extended TA features to properly mimic the time-enriched LSC dynamics; and consequently
- to exploit the power of the UPPAAL model checker for scenario-based automatic verification of non-trivial real-time systems.

The proposed first approach has been implemented as a new feature inside UPPAAL, and the second approach implemented as an LSC-to-TA translator which, together with our LSC editor and UPPAAL, constitutes a tool chain for scenario-based verification. Since the translation, the composition and the underlying verification are all automatic steps, our methods are fully automated. Preliminary experiments with a Train-Gate system indicate that the proposed approaches are computationally feasible and effective.

The benefits of building our scenario-based real-time system verification methods on top of the well-developed real-time model checker are twofold: (1) for system analysts and designers who are interested in early-stage validations using live sequence charts, now they can scale up to the *timed* settings without having to develop and implement the corresponding real-time verification algorithms; (2) for users of conventional (real-time) model checkers that work with state/transition-based models and temporal logical properties, now they can horizontally scale up to *scenario-based* system models and *scenario-based* user requirements.

In our proposed approaches, the time-enriched LSC chart in its current form has only a limited number of language constructs. To ease the scenario-based characterization of practically relevant complex systems using LSC and thus to realize the full potential of scenario-based approaches, we need to support more language constructs such as subchart, if-then-else structure, loop, forbidden and ignored messages, coregion, and symbolic messages and instances. Accordingly, we need to implement the full-fledged translators, and to apply the tool and tool chain to larger (industrial) case studies. Another limitation of the current approaches is that counterexamples (if any) can be displayed only in the translated timed automata. For the users' convenience, it will be extremely useful to trace the counterexamples back into the LSC system models and the LSC requirements. This needs to be achieved in the near future.

Acknowledgements We would like to thank the anonymous referees for their constructive comments and helpful suggestions. Thanks are also due to Itai Segall for pointing out an error in an earlier version of the "one-TA-per-instance line" translation.

References

1. Luca Aceto, Augusto Burgueño, and Kim Guldstrand Larsen. Model checking via reachability testing for timed automata. In *Proc. 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'98)*, pages 263–280, 1998.
2. Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
3. Rajeev Alur and Thomas A. Henzinger. Real-time system = discrete system + clock variables. *Software Tools for Technology Transfer (STTT)*, 1(1-2):86–109, 1997.
4. Rajeev Alur, Gerard J. Holzmann, and Doron Peled. An analyzer for message sequence charts. *Software - Concepts and Tools*, 17(2):70–77, 1996.
5. Sandie Balaguer. Specification of properties using live sequence charts - theory and implementation. Master's thesis, École Centrale de Nantes, Nantes, France, September 2009.
6. Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini, editors, *SFM*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004.
7. Yves Bontemps. *Relating Inter-Agent and Intra-Agent Specifications - The Case of Live Sequence Charts*. PhD thesis, University of Namur, Namur, Belgium, 2005.
8. Yves Bontemps and Pierre-Yves Schobbens. The computational complexity of scenario-based agent verification and design. *J. Applied Logic*, 5(2):252–276, 2007.
9. Annette Bunker, Ganesh Gopalakrishnan, and Konrad Slind. Live sequence charts applied to hardware requirements specification and verification. *Software Tools for Technology Transfer (STTT)*, 7(4):341–350, 2005.
10. Pierre Combes, David Harel, and Hillel Kugler. Modeling and verification of a telecommunication application using live sequence charts and the play-engine tool. *Software and System Modeling*, 7(2):157–175, 2008.
11. Werner Damm and David Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.
12. Werner Damm and Jochen Klose. Verification of a radio-based signaling system using the statemate verification environment. *Formal Methods in System Design*, 19(2):121–141, 2001.
13. Werner Damm, Tobe Toben, and Bernd Westphal. On the expressive power of live sequence charts. In *Program Analysis and Compilation, Theory and Practice, Essays Dedicated to Reinhard Wilhelm on the Occasion of His 60th Birthday*, pages 225–246, 2006.
14. Thomas Firley, Michaela Huhn, Karsten Diethers, Thomas Gehrke, and Ursula Goltz. Timed sequence diagrams and tool-based analysis - a case study. In *Proc. 2nd International Conference on the Unified Modeling Language (UML'99)*, pages 645–660, 1999.
15. Blaise Genest, Marius Minea, Anca Muscholl, and Doron Peled. Specifying and verifying partial order properties using template MSCs. In *Proc. 7th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'04)*, pages 195–210, 2004.
16. David Harel and Hillel Kugler. Synthesizing state-based object systems from LSC specifications. *Int. J. Found. Comput. Sci.*, 13(1):5–51, 2002.
17. David Harel, Hillel Kugler, Rami Marelly, and Amir Pnueli. Smart play-out of behavioral requirements. In *Proc. 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD'02)*, pages 378–398, 2002.
18. David Harel and Rami Marelly. *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.
19. Klaus Havelund, Kim Guldstrand Larsen, and Arne Skou. Formal verification of a power controller using the real-time model checker UPPAAL. In *Proc. 5th International AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems (ARTS'99)*, pages 277–298, 1999.

20. ITU-T. Message Sequence Charts – MSC-2000, ITU-T recommendation z.120, 1999.
21. Mohammad Mahdi Jaghoori and Tom Chothia. Timed automata semantics for analyzing Creol. *CoRR*, abs/1007.5095, 2010.
22. Mohammad Mahdi Jaghoori, Frank S. de Boer, Tom Chothia, and Marjan Sirjani. Schedulability of asynchronous real-time concurrent objects. *J. Log. Algebr. Program.*, 78(5):402–416, 2009.
23. Jochen Klose. *Live Sequence Charts: A Graphical Formalism for the Specification of Communication Behavior*. PhD thesis, Carl von Ossietzky Universität Oldenburg, 2003.
24. Jochen Klose, Tobe Toben, Bernd Westphal, and Hartmut Wittke. Check it out: On the efficient formal verification of live sequence charts. In *Proc. 18th International Conference on Computer Aided Verification (CAV'06)*, pages 219–233, 2006.
25. Jochen Klose and Hartmut Wittke. An automata based interpretation of live sequence charts. In *Proc. 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, pages 512–527, 2001.
26. Hillel Kugler, David Harel, Amir Pnueli, Yuan Lu, and Yves Bontemps. Temporal logic for scenario-based specifications. In *Proc. 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05)*, pages 445–460, 2005.
27. Jussi Lahtinen. Model checking timed safety instrumented systems. Master's thesis, Helsinki University of Technology, Espoo, Finland, June 2008. Research Report TKK-ICS-R3.
28. Kim Guldstrand Larsen, Shuhao Li, Brian Nielsen, and Saulius Pusinskas. Verifying real-time systems against scenario-based requirements. In *Proc. 16th Int'l Symposium on Formal Methods (FM'09)*, pages 676–691, 2009.
29. Kim Guldstrand Larsen, Shuhao Li, Brian Nielsen, and Saulius Pusinskas. Scenario-based analysis and synthesis of real-time systems using UPPAAL. In *Proc. 13th Conference on Design, Automation, and Test in Europe (DATE'10)*, pages 447–452, 2010.
30. Kim Guldstrand Larsen, Shuhao Li, Brian Nielsen, and Saulius Pusinskas. Scenario-based verification of real-time systems using UPPAAL. Technical report, Aalborg University Department of Computer Science, Aalborg, Denmark, 2010, available `http://www.cs.aau.dk/~li/papers/ScenarioVeriUppaalTR.pdf`.
31. Marc Lettrari and Jochen Klose. Scenario-based monitoring and testing of real-time UML models. In *Proc. 4th International Conference on the Unified Modeling Language (UML'01)*, pages 317–328, 2001.
32. Kuchi V. S. Prasad. A calculus of broadcasting systems. *Sci. Comput. Program.*, 25(2-3):285–327, 1995.
33. Saulius Pusinskas. *Capturing and Testing Behavioral Requirements by Means of Live Sequence Charts*. PhD thesis, Aalborg University, Aalborg, Denmark, 2010.
34. Jens Gorm Rye-Andersen, Mads Jensen, Rene Goettler, and Michael Jakobsen. PEEL: Property Extraction Engine for LSCs. Master's thesis, Aalborg University, Aalborg, Denmark, 2004.
35. Ingo Schinz, Tobe Toben, Christian Mrugalla, and Bernd Westphal. The Rhapsody UML verification environment. In *Proc. 2nd International Conference on Software Engineering and Formal Methods (SEFM'04)*, pages 174–183, 2004.
36. Bikram Sengupta and Rance Cleaveland. Triggered message sequence charts. In *Proc. 10th ACM SIGSOFT Symposium on Foundations of Software Engineering (SIGSOFT FSE'02)*, pages 167–176, 2002.
37. Jun Sun and Jin Song Dong. Model checking live sequence charts. In *Proc. 10th International Conference on Engineering of Complex Computer Systems (ICECCS'05)*, pages 529–538, 2005.
38. The UPPAAL Team. UPPAAL 4.0.8 online help document. Available at `http://www.uppaal.com`, 2009.
39. Tao Wang, Abhik Roychoudhury, Roland H. C. Yap, and Shishir C. Choudhary. Symbolic execution of behavioral requirements. In *Proc. 6th International on Practical Aspects of Declarative Languages (PADL'04)*, pages 178–192, 2004.
40. Wang Yi, Paul Pettersson, and Mats Daniels. Automatic verification of real-time communicating systems by constraint-solving. In *Proc. 7th IFIP WG6.1 International Conference on Formal Description Techniques (FORTE'95)*, pages 243–258, 1994.
41. Sergio Yovine. KRONOS: A verification tool for real-time systems. *Software Tools for Technology Transfer (STTT)*, 1(1-2):123–133, 1997.

A BNF grammar of time-enriched LSC

The following BNF grammar describes our time-enriched LSC language.

```
LSC ::= TYPE MODE INSTANCES CHART
TYPE ::= type (existential | universal)
MODE ::= mode (initial | iterative | invariant)
INSTANCES ::= INSTANCES INSTANCES
| instance INST_ID NAME
CHART ::= chartbegin CHARTBODY chartend
CHARTBODY ::= CHARTBODY CHARTBODY
| message ELEM_ID INST_ID INST_ID YLOCATION NAME
| condition ELEM_ID (INST_ID)+ YLOCATION TEMP EXPR
| assignment ELEM_ID (INST_ID)+ YLOCATION UPDATE
| simregion ELEM_ID (INST_ID)+ YLOCATION
| pchbot ELEM_ID (INST_ID)+ YLOCATION
INST_ID ::= <numeral>
ELEM_ID ::= <numeral>
YLOCATION ::= <numeral>
TEMP ::= cold | hot
EXPR ::= <boolean expression>
UPDATE ::= <clock resets>
```

As can be seen from the BNF grammar, each primitive construct (i.e., message, condition, assignment) has an element ID and a y-coordinate. This y-coordinate denotes the geographical distance from the element to the top of all instance lines (note that all messages, conditions and assignments have horizontal layouts). It should not be confused with the numbering of a "position" among all the points of communication, computation and synchronization along an instance line.

In the BNF grammar, a simregion is represented by the y-coordinate where its constituent message, and/or condition, and/or assignment are anchored together (Fig. 5, black filled circles).

Furthermore, a prechart is represented by its bottom y-coordinate vector that spans across the relevant instance lines. When there is no prechart, the "pchbot" statement will not appear in the LSC file.

B Timed automata in UPPAAL

We use the following notations: X is a set of real-valued clocks, and $B(X)$ is the set of conjunctions over simple conditions of the form $x \bowtie c$ or $x - y \bowtie c$, where $x, y \in X$, $c \in \mathbb{N}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$.

Definition 9 (timed automaton, TA [6]) A *timed automaton* is a tuple (L, l_0, X, Act, E, Inv), where L is a set of locations, $l_0 \in L$ is the initial location, X is a set of clocks, Act is the alphabet of actions, $E \subseteq L \times (Act \cup \{\tau\}) \times B(X) \times 2^X \times L$ is a set of edges between locations, each of which has an action, a guard and a set of clocks to be reset, and $Inv : L \to B(X)$ assigns invariants to locations. □

UPPAAL has defined a number of extensions to the standard notations of timed automata [2]. Specifically, an *urgent* location is such a TA location that freezes time, i.e., time is not allowed to elapse when a process is in an urgent location. A *committed* location is a special kind of urgent location whose outgoing transitions always have higher priority to be fired than those from non-committed locations.

UPPAAL uses a mixture of handshake communication and broadcast communication. The CBS (Calculus of Broadcasting Systems [32])-style broadcast channels allow one-to-many synchronization. If the emitting edge is enabled, then it can always fire. If the emitting edge is to fire, then all enabled receiving edges (might be 0 edge) will synchronize.

In UPPAAL an *urgent* channel means that if it is possible to trigger a synchronization over that channel, then it cannot delay in the source state.

Furthermore, UPPAAL also supports bounded-range integer and boolean data variables, which can be used in the guards, assignment and location invariants.

A clock valuation is a function $u : X \to \mathbb{R}_{\geq 0}$ that assigns each clock variable a non-negative real number. Let ${\mathbb{R}_{\geq 0}}^X$ be the set of all clock valuations. Let $u_0(x) = 0$ for all $x \in X$. We may consider guards and invariants as sets of clock valuations. For example, we use $u \in Inv(l)$ to denote that valuation u satisfies $Inv(l)$.

Definition 10 (semantics of TA [6]) Let (L, l_0, X, Act, E, Inv) be a timed automaton. The *semantics* is defined as a labeled transition system $\langle SS, s_0, \to\rangle$, where $SS \subseteq L \times {\mathbb{R}_{\geq 0}}^X$ is the set of semantic states, $s_0 = (l_0, u_0) \in SS$ the initial state, and $\to \subseteq SS \times (Act \cup \{\tau\} \cup \mathbb{R}_{\geq 0}) \times SS$ the transition relation such that:

- $(l,u) \xrightarrow{d} (l, u+d)$ if $\forall d' : 0 \leq d' \leq d \,.\, u+d' \in Inv(l)$; and
- $(l,u) \xrightarrow{a} (l', u')$ if there exists $e = (l, a, g, r, l') \in E$ such that $u \in g$, $u' = [r \to 0]u$, and $u' \in Inv(l')$,

where for $d \in \mathbb{R}_{\geq 0}$, $u+d$ maps each clock x in X to the value $u(x)+d$, and $[r \to 0]u$ denotes the clock valuation which maps each clock in r to 0 and agrees with u over $X \backslash r$. □

Definition 11 (run of TA) A *run* of a TA (L, l_0, X, Act, E, Inv) is a sequence of states $s^0 \cdot s^1 \cdot \ldots$ that are connected by the transitions, i.e., $\forall i \geq 0 \,.\, \exists u_i \in (Act \cup \{\tau\} \cup \mathbb{R}_{\geq 0}) \,.\, s^i \xrightarrow{u_i} s^{i+1}$. □

The transition relation $\to$ as mentioned above each time consumes only a single letter $u \in (Act \cup \{\tau\} \cup \mathbb{R}_{\geq 0})$. We extend it to $\to^*$ such that it consumes a (finite or infinite) word $w \in (Act \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^* \cup (Act \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^\omega$. A word w that corresponds to a run of the TA is called a *timed trace* of the TA.

A number of timed automata can be composed in parallel into a network of timed automata over a common set of clocks and actions, $A_i = (L_i, l_{0,i}, X, Act, E_i, Inv_i)$, $1 \leq i \leq n$. A location vector $\bar{l} = (l_1, \ldots, l_n)$ is a vector of locations of the member TAs. We compose the invariant functions into a common function over location vectors $Inv(\bar{l}) = \bigwedge_i Inv_i(l_i)$. We write $\bar{l}[l'_i/l_i]$ to denote the vector where the i-th element l_i of $\bar{l}$ is replaced by l'_i.

Definition 12 (semantics of a network of TAs [6]) Let $A_i = (L_i, l_{0,i}, X, Act, E_i, Inv_i)$ be a network of timed automata, $1 \leq i \leq n$. Let $\bar{l_0} = (l_{0,1}, \ldots, l_{0,n})$ be the initial location vector. The *semantics* is defined as a transition system $\langle SS, s_0, \rightarrow \rangle$, where $SS = (L_1 \times \ldots \times L_n) \times \mathbb{R}_{\geq 0}{}^X$ is the set of global semantic states, $s_0 = (\bar{l_0}, u_0) \in SS$ the initial global state, and $\rightarrow \subseteq SS \times (Act \cup \{\tau\} \cup \mathbb{R}_{\geq 0}) \times SS$ the transition relation defined by:

- $(\bar{l}, u) \xrightarrow{d} (\bar{l}, u+d)$ if $\forall d' : 0 \leq d' \leq d \,.\, u + d' \in Inv(\bar{l})$;
- $(\bar{l}, u) \xrightarrow{\tau} (\bar{l}[l'_i/l_i], u')$ if there exists $l_i \xrightarrow{\tau, g, r} l'_i$ such that $u \in g$, $u' = [r \rightarrow 0]u$ and $u' \in Inv(\bar{l}[l'_i/l_i])$;
- $(\bar{l}, u) \xrightarrow{a} (\bar{l}[l'_i/l_i, l'_j/l_j], u')$ if a is a binary channel and there exist $l_i \xrightarrow{c!, g_i, r_i} l'_i$ and $l_j \xrightarrow{c?, g_j, r_j} l'_j$ such that $u \in (g_i \wedge g_j)$, $u' = [r_i \cup r_j \rightarrow 0]u$ and $u' \in Inv(\bar{l}[l'_i/l_i, l'_j/l_j])$; and
- $(\bar{l}, u) \xrightarrow{a} (\bar{l}[l'_i/l_i, l'_j/l_j, l'_k/l_k, \ldots], u')$ if a is a broadcast channel and there exist an $l_i \xrightarrow{c!, g_i, r_i} l'_i$ and a maximal set $\{j, k, \ldots\}$: $l_j \xrightarrow{c?, g_j, r_j} l'_j$, $l_k \xrightarrow{c?, g_k, r_k} l'_k$, ..., such that $u \in (g_i \wedge g_j \wedge g_k \wedge \ldots)$, $u' = [r_i \cup r_j \cup r_k \cup \ldots \rightarrow 0]u$ and $u' \in Inv(\bar{l}[l'_i/l_i, l'_j/l_j, l'_k/l_k, \ldots])$.

□

Runs and traces of a network of TAs are defined similarly as those for a single TA.

C Proofs of lemmas and theorems in Section 4

Theorem 1 If a configuration (c, v) of L corresponds to a semantic state (l, v) of O_L, then: (1) each simregion s that follows (c, v) in L uniquely corresponds to an outgoing edge (l, l') in O_L; and (2) the target configuration (c', v') of s in L uniquely corresponds to the target semantic state (l', v') in O_L.

Proof For each simregion s in L that immediately follows (c, v), according to Section 4.1.3, s uniquely corresponds to an outgoing edge (l, l') from l in O_L. Since the valuation function v is the same in (l, v) as in (c, v), and the condition in s is straightforward copied onto the TA edge (l, l'), the simregion s can be stepped over if and only if the TA edge (l, l') can be taken. Moreover, the assignment (if any) in s is also straightforward copied onto the edge (l, l'). This indicates that the valuation function in the LSC target configuration will be still the same as in the TA target semantic state. Therefore, (c', v') uniquely corresponds to (l', v').

Specifically, if s is a non-message simregion that immediately follows (c, v) in L, then according to the ASAP semantics, s will be stepped over immediately from (c, v). Accordingly, the source location l is a committed location in O_L, and the other outgoing edges that correspond to message simregions will not be appended to l. All these ensure that the TA edge that corresponds to s is taken immediately from state (l, v). □

If in the LSC semantics (Section 3.2.1) we ignore the silent steps that are caused by intra-chart coordinations and autonomous advancements of instance lines, then we have:

Theorem 2 For any trace *tr* in O_L: $tr \models L \Leftrightarrow (O_L, tr) \models (l_{min} \rightsquigarrow l_{max})$.

Proof Let the initial cut of L be c_0. According to Section 4.1.2, c_0 corresponds to the initial location l_0 of O_L. Since in the beginning all the clocks in L have the same initial values as in O_L, the initial configuration (c_0, v_0) of L uniquely corresponds to the initial semantic state (l_0, v_0) of O_L.

We consider only the legal (admissible) behaviors of O_L. In other words, the traces that lead to the sink TA location Err will be ignored. We consider the following cases:

(1) O_L has only explicitly specified behaviors. By Theorem 1, each simregion that immediately follows (c_0, v_0) uniquely corresponds to an outgoing edge from TA location l_0, and the target configuration (c', v') in L uniquely corresponds to the target semantic state (l', v') in O_L. On the other hand, in (c_0, v_0) of L, there could be a time delay $d \in \mathbb{R}_{\geq 0}$ if and only if in (l_0, v_0) of O_L there could be the same time delay d. By recursively applying Theorem 1 and the above result, we can conclude that any timed trace *tr* in O_L is also a timed trace in L.

By assuming that O_L has only explicitly specified behaviors, we know that there is no undesired behavior in O_L. If $tr \models L$, then by definition this particular *tr* in O_L also satisfies the path formula $(l_{min} \rightsquigarrow l_{max})$, i.e., $(O_L, tr) \models (l_{min} \rightsquigarrow l_{max})$. Therefore, we have $tr \models L \Rightarrow (O_L, tr) \models (l_{min} \rightsquigarrow l_{max})$.

The reverse implication is proved similarly.

(2) O_L includes behaviors of unconstrained events or cold violations. In this case, each unconstrained event m at a particular cut c in L uniquely corresponds to an m?-labeled self-loop edge at the corresponding location l in O_L, and each cold violation uniquely corresponds to an edge leading to l_{pmin}. The two-way implications are proved similarly.

(3) O_L includes behaviors of prechart pre-matching. In this case, the semantics of $tr \models L$ says that whenever *tr* matches the prechart *Pch*, the main chart *Mch* will be matched afterwards (and this must happen before *Pch* begins a next round matching). Considering that in O_L, the locations l_{min} and l_{max} are two rendezvous points, thus $tr \models L$ means exactly the satisfaction of $(l_{min} \rightsquigarrow l_{max})$ by *tr*.

To sum up, we conclude that for any trace *tr* in O_L, we have $tr \models L \Leftrightarrow (O_L, tr) \models (l_{min} \rightsquigarrow l_{max})$. □

Let the modified version of the original system model S be S', and the modified version of the observer timed automaton O_L for chart L be O'_L. Let the minimal and maximal cuts of the main chart of L correspond to locations l_{min} and l_{max} of O'_L, respectively. When L is a universal chart, we have:

Lemma 1 *If O_L has no committed location, and all $ch \in \Pi$ are binary synchronization channels, then $S \models L \Leftrightarrow (S' \,||\, O'_L) \models (l_{min} \rightsquigarrow l_{max})$.* □

Proof Let $(\bar{l}, v)$ be a semantic state of the network of TAs of S, where $\bar{l}$ is a location vector, and v is the valuation of all clock variables. According to the UPPAAL semantics on binary synchronizations [38], for each binary synchronization channel $ch \in \Pi$, we have a transition $(\bar{l}, v) \xrightarrow{ch} (\bar{l}', v')$ if in two different processes (TAs) of S, there are two edges (l_i, l_{i+1}) and (l_j, l_{j+1}) labeled with $\mathsf{ch!}$ and $\mathsf{ch?}$, respectively, such that:

- $v \models g_i \wedge g_j$, where g_i and g_j are guards of the two edges, respectively;
- $\bar{l}' = \bar{l}[l_{i+1}/l_i, l_{j+1}/l_j]$;
- $v' = a_j(a_i(v))$, where a_i and a_j are the assignments of the emitting and receiving edges, respectively;
- $v' \models Inv_{i+1} \wedge Inv_{j+1}$, where Inv_{i+1} and Inv_{j+1} are the location invariants of the target locations of the two edges, respectively;
- either (l_i or l_j or both are committed locations), or no other location in $\bar{l}$ is committed.

We need to show that the modifications of the original system model S and the observer TA O_L do not affect their legal (i.e. admissible) behaviors, i.e., the event notification mechanism and the locking mechanisms neither increase nor decrease the behaviors (traces) in S and O_L. To this end, we prove that each synchronization in S uniquely corresponds to a pair of consecutive synchronizations in ($S' \,||\, O'_L$).

⇒)

By $S \models L$ we know that the original system model S satisfies the requirements that are specified in the LSC chart L. It follows that the observer TA O_L does not restrict the (legal) behaviors of S.

If at a semantic state $(\bar{l}, v)$ of S there is a synchronization $(\bar{l}, v) \xrightarrow{ch} (\bar{l}', v')$, where $ch \in \Pi$, we let the two coupling edges that carry $\mathsf{ch!}$ and $\mathsf{ch?}$ be (l_i, l_{i+1}) and (l_j, l_{j+1}), respectively. Clearly, they satisfy all the five requirements as listed earlier in this proof. According to the rules for modifying S, the edge (l_i, l_{i+1}) in S will correspond to two edges (l_i, l'_i) and (l'_i, l_{i+1}) in S', where l'_i is a newly added committed location. Also according to the modification rules, the semaphore *mayFire* evaluates to false only when the current control is in a newly added committed location (Fig. 13(a)). Now that the control is in l_i in S', the semaphore *mayFire* should evaluate to true. This together with "$v \models g_i \wedge g_j$" (the first item requirement) indicates that the guards for the edges (l_i, l'_i) and (l_j, l_{j+1}) of S' to synchronize on channel ch are both satisfied. Besides, items 3-5 in the binary synchronization requirements also apply to the ch-synchronization at (l_i, l'_i) and (l_j, l_{j+1}). Therefore, there exists a transition $(\bar{l}, v) \xrightarrow{ch} (\bar{l}'', v')$ in S' with (l_i, l'_i) and (l_j, l_{j+1}) as the coupling edges, where $\bar{l}'' = \bar{l}'[l'_i/l_i, l_{j+1}/l_j]$.

The second edge (l'_i, l_{i+1}) in S' will be immediately coupled with a corresponding edge in O'_L. By the assumption $S \models L$, we know that O_L does not restrict the behaviors of S via its own conditions (e.g., via g_3 in Fig. 13(b)). This means that the cho-synchronization between S' and O'_L will not get stuck there due to the restrictions of O'_L. Since after this synchronization, the clock variables in S' remain unchanged, we know that the location invariant Inv_{i+1} on l_{i+1} of S' will still be satisfied. After this synchronization, the two target locations in S' will be l_{i+1} and l_{j+1}, thus coinciding with the corresponding target locations l_{i+1} and l_{j+1} in S. Therefore, we can conclude that given a trace tr in S, there exists a unique trace tr' in $(S' \,||\, O'_L)$ such that tr' and tr correspond.

By the definition of $S \models L$ (see Section 3.2), we know that if a timed trace μ in S arrives at the minimal cut of the main chart of L, then μ must always be able to reach the maximal cut of that main chart. By Theorem 2 and Section 4.1, we know that if μ arrives at location l_{min} of O'_L, then μ must always be able to reach location l_{max} of O'_L.

Since each trace μ in S can be equivalently mapped to a trace μ' in $(S' \,||\, O'_L)$, clearly, if any μ' arrives at location l_{min} of O'_L, then that μ' must always be able to reach location l_{max} of O'_L.

Since l_{min} and l_{max} are two locations in $(S' \,||\, O'_L)$, the above requirement can thus be formulated as a UPPAAL property $(S' \,||\, O'_L) \models (l_{min} \leadsto l_{max})$.

$\Leftarrow$)

We need to prove that each trace tr' in $(S' \,||\, O'_L)$ that satisfies the CTL property uniquely corresponds to a trace tr in S that satisfies the LSC requirement.

Assume that in $(S' \,||\, O'_L)$ there is a synchronization $(\bar{l}, v) \xrightarrow{c} (\bar{l}', v')$.

If $c \in \Pi$, then after removing "*mayFire* == true" from the condition and removing "*mayFire* := false" from the assignment of the emitting edge (Fig. 13(a)), the edge becomes exactly the corresponding edge in S. Note that the invariant (if any) at the target location of this emitting edge is irrelevant of the semaphore *mayFire*. This indicates that the synchronization between the corresponding edges in S can also fire.

If c is a fresh channel (i.e., in the form of cho), then the source location of the c!-emitting edge in S' must be a newly added committed location. This c! will be synchronized with a c?-receiving edge in O'_L. And it will bring the control in S' from the committed location to the target location, which coincides with the corresponding target location in S. Due to the use of semaphore *mayFire*, no other synchronizations in $(S' \,||\, O'_L)$ can preempt the execution of this c-synchronization.

The rest of the transitions in $(S' \,||\, O'_L)$ are just the same as those in S. Therefore we can conclude that a trace tr' in $(S' \,||\, O'_L)$ uniquely corresponds to a trace tr in S such that tr' and tr are equivalent. Now that $(S' \,||\, O'_L) \models (l_{min} \leadsto l_{max})$, according to the semantics of LSC chart satisfaction, we have $S \models L$. □

Let S, L, O_L, S', O'_L, l_{min} and l_{max} be the same as declared and explained in Lemma 1. When L is a universal chart, we have:

Theorem 3 $S \models L \Leftrightarrow (S' \,||\, O'_L) \models (l_{min} \leadsto l_{max})$.

Proof This theorem is a generalization of Lemma 1 by canceling the restrictions.

If $ch \in \Pi$ is a broadcast channel, the semantics of ch-synchronization [38] is a little different. Since the modifications of the emitting edges in S do not affect the receiving edges in S, we can still have a one-to-one mapping between the traces in S and in $(S' \,||\, O'_L)$.

If there are committed locations in O'_L, then we use the second semaphore *NxtCmt* to guarantee the non-interrupted execution at those committed locations in O'_L. Since an edge (l, l') starting from a committed location l in O'_L represents an internal action

(τ) transition (i.e., a local transition), it needs no synchronization with S'. Thus the edge does not affect the behavior of S'.

To sum up, there is a one-to-one mapping of the traces in S and in $(S' \,||\, O'_L)$, even in the presences of broadcast channels in S and committed locations in O_L. Thus we have $S \models L \Leftrightarrow (S' \,||\, O'_L) \models (l_{min} \rightsquigarrow l_{max})$. □

Let S, L, O_L, S', O'_L and l_{max} be the same as explained in Theorem 3. When L is an existential chart, we have:

Theorem 4 $S \models L \Leftrightarrow (S' \,||\, O'_L) \models \mathsf{E}\Diamond\, l_{max}$.

Proof (idea) We can show that there is a one-to-one mapping between the traces in S and in $(S' \,||\, O'_L)$ similarly as in Lemma 1 and Theorem 3.

The main difference between this theorem and Theorem 3 lies in the semantics of a universal chart and of an existential chart. However, it is clear that the CTL formula $\mathsf{E}\Diamond\, l_{max}$ represents exactly the existential chart requirements. So this theorem can be proved similarly as in Lemma 1 and Theorem 3. □

D Proofs of lemmas and theorems in Section 5

Let L be an untimed LSC chart whose instance lines $I_1, I_2, \ldots, I_n$ correspond to timed automata $A_1, A_2, \ldots, A_n$, respectively, then the translated network of TAs will be $NTA_L = \{A_i \mid 1 \leq i \leq n\} \cup \{Coord\}$. According to rules R3, R4 and R6, there will be a set of auxiliary channels $Aux = \{pch\_over_i, mch\_over_i \mid 1 \leq i \leq n\} \cup \{activate, over, pch\_vio, reset\}$ that will be used in NTA_L. Let the message alphabet of L be Σ, then the alphabet of observable actions in NTA_L will be $Act = (\Sigma \cup Aux)$.

Lemma 2 *Let L be an untimed LSC chart whose message alphabet is Σ, and let NTA_L be the translated network of timed automata which have a set Act of observable actions. Then* $\forall \gamma_1 \in (\Sigma \cup \{\tau\})^{\omega} \,.\, ((\gamma_1 \models L) \Rightarrow \exists \gamma_2 \in (Act \cup \{\tau\})^{\omega} \,.(\gamma_2 \models NTA_L) \wedge (\gamma_2|_{\Sigma} = \gamma_1|_{\Sigma}))$*, and* $\forall \gamma_2 \in (Act \cup \{\tau\})^{\omega} \,.\, ((\gamma_2 \models NTA_L) \Rightarrow \exists! \gamma_1 \in (\Sigma \cup \{\tau\})^{\omega} \,.\, (\gamma_1 \models L) \wedge (\gamma_2|_{\Sigma} = \gamma_1|_{\Sigma}))$.

Proof We can prove the above two implications by proving that each cut of chart L uniquely corresponds to a location vector in the network of timed automata NTA_L, and each advancement step in L uniquely corresponds to either a message synchronization transition (ranging on $\Sigma \cup Aux$) or a sequence of concatenated message synchronization and internal action transitions in NTA_L, such that they consume exactly the same letter from Σ if they are both restricted to Σ. Note that we restrict the LSC advancement steps to represent only legal (i.e. admissible) behaviors.

Let the instance lines in chart L be $I_1, I_2, \ldots, I_n$. They will be translated into timed automata $A_1, A_2, \ldots, A_n$, respectively. Together with the auxiliary coordinator automaton $Coord$ they constitute NTA_L.

The initial cut c^0 of chart L corresponds to the LSC initial position vector $(0_1, 0_2, \ldots, 0_n)$, where i_j means that instance $I_j \in inst(L)$ is currently in its position $i \in pos(L, I_j)$. In the translated network of timed automata NTA_L, automaton $Coord$ is initially in its location l^0_{coord}. By rule R1, each 0_i in position vector $(0_1, 0_2, \ldots, 0_n)$ corresponds to a TA location l^0_i (denoting location 0 in timed automaton A_i). Therefore, cut c_0 uniquely corresponds to the NTA_L initial location vector $\bar{l}^0 = (l^0_1, l^0_2, \ldots, l^0_n, l^0_{coord})$.

We show how the advancement steps from the LSC initial position vector correspond to the transitions in the network of timed automata. At LSC position vector $(0_1, 0_2, \ldots, 0_n)$, there are two kinds of possible advancement steps:

- If there is an m-labeled message occurrence mo from position 1_i of instance I_i to position 1_j of instance I_j (i.e., mo is a minimal event), then:
  On one hand, by rule R2, there will be an $m!$-labeled TA edge from location l^0_i to l^1_i in A_i, and an $m?$-labeled TA edge from location l^0_j to l^1_j in A_j. According to the LSC semantics, there is a message synchronization advancement step on m in L from $(0_1, \ldots, 0_i, \ldots, 0_j, \ldots, 0_n)$ to $(0_1, \ldots, 1_i, \ldots, 1_j, \ldots, 0_n)$. Accordingly, in NTA_L there exists exactly a corresponding binary synchronization on channel m between A_i and A_j, and the location vector of NTA_L will change from $(l^0_1, \ldots, l^0_i, \ldots, l^0_j, \ldots, l^0_n, l^0_{coord})$ to $(l^0_1, \ldots, l^1_i, \ldots, l^1_j, \ldots, l^0_n, l^0_{coord})$.
  On the other hand, according to the semantics of the invariant mode universal chart, the message as a minimal event can be constantly matched for with L staying in the initial cut. By rule R8, in NTA_L there will be first a binary synchronization on channel m, i.e., $(l^0_1, \ldots, l^0_i, \ldots, l^0_j, \ldots, l^0_n, l^0_{coord}) \xrightarrow{m} (l^0_1, \ldots, l^0_i, \ldots, l^{PM}_j, \ldots, l^0_n, l^0_{coord})$, and then an immediately following internal action transition that leads back to the initial location vector, i.e., $(l^0_1, \ldots, l^0_i, \ldots, l^{PM}_j, \ldots, l^0_n, l^0_{coord}) \xrightarrow{\tau} (l^0_1, \ldots, l^0_i, \ldots, l^0_j, \ldots, l^0_n, l^0_{coord})$. Here l^{PM}_j is an auxiliary TA location that is specially used for prechart pre-matching. In this case of pre-matching, the m-synchronization advancement step in L uniquely corresponds to a sequence of the tightly concatenated $\xrightarrow{m}$ and $\xrightarrow{\tau}$ transitions.
  Since a dedicated flag boolean variable *prematch* has been used to strengthen the TA transition guards, assignments and the location invariants, it follows that at NTA_L location vector $(l^0_1, \ldots, l^0_i, \ldots, l^0_j, \ldots, l^0_n, l^0_{coord})$, there are only the two above-mentioned possible interleaved executions between the two $m!$-labeled outgoing edges from l^0_i in I_i and the two $m?$-labeled outgoing edges from l^0_j in I_j.
- If instance I_i has no interactions with other instance lines in the prechart, then there is an immediate silent advancement step from $(0_1, \ldots, 0_i, \ldots, 0_n)$ to $(0_1, \ldots, 1_i, \ldots, 0_n)$. By rule R3, l^0_i will be a committed location in A_i of NTA_L, and there will be a *pch_over*$_i$!-labeled edge from l^0_i to l^1_i. Furthermore, in automaton *Coord* there will be a coupling *pch_over*$_i$?-labeled edge either
  - from l^0_{coord} to l^1_{coord}, corresponding to the case where I_i is the very last instance line of L to complete its prechart portion; or
  - from l^0_{coord} to l^0_{coord}, corresponding to the case where I_i is not yet the last instance line of L to complete its prechart portion.

  In the two cases, the location vector of NTA_L will be changed from $(l^0_1, \ldots, l^0_i, \ldots, l^0_n, l^0_{coord})$ to $(l^0_1, \ldots, l^1_i, \ldots, l^0_n, l^1_{coord})$, and from $(l^0_1, \ldots, l^0_i, \ldots, l^0_n, l^0_{coord})$ to $(l^0_1, \ldots, l^1_i, \ldots, l^0_n, l^0_{coord})$, respectively. However, in both cases, there will be exactly one binary synchronization transition on *pch_over*$_i$ in NTA_L.

The above two kinds of possible advancement steps indicate that there is an initial correspondence between the position vector of L and the location vector of NTA_L.[3] Since an untimed chart is a message-only chart, a cut vector is itself an LSC configuration, and a location vector is itself a semantic state of the translated network of timed automata.[4] Therefore, there is an initial "LSC cut to TA location vector", and "LSC advancement step to TA (sequence of) transition" correspondence between L and NTA_L.

The above correspondences can be generalized by using induction. Assume that at a cut c that corresponds to a position vector $(p1_1, \ldots, pi_i, \ldots, pj_j, \ldots, pn_n)$ in the prechart of L, there is an m-labeled message occurrence sent from position $(pi+1)_i$ of instance I_i to position $(pj+1)_j$ of instance I_j. If for cut c, there uniquely exists a corresponding location vector $\bar{l}$ in NTA_L, then similar to the case of the initial cut, we can prove that the message synchronization advancement step on m in L uniquely corresponds to a binary synchronization transition in NTA_L; and after this message synchronization advancement step, the new cut c' uniquely corresponds to the destination location vector $\bar{l}'$ in NTA_L. Proof by induction ensures that any normal (i.e., other than the prechart pre-matching ones) message synchronization advancement step in the prechart of L uniquely corresponds to a message synchronization transition in NTA_L.

When $(p1_1, \ldots, pi_i, \ldots, pj_j, \ldots, pn_n)$ is a position vector in the main chart of L, the unique correspondence relation can be proved similarly.

Now we prove the unique correspondence for the case that involves the intra-chart coordination (e.g., the prechart to main chart transition). Assume that in the prechart of L, a cut c corresponds to position vector $(p1_1, \ldots, pi_i, \ldots, pn_n)$, where $pi + 1 = Pch\_bot_{L,I_i}$. If $(p1_1, \ldots, pi_i, \ldots, pn_n)$ uniquely corresponds to a location vector $(l_1^{p1}, \ldots, l_i^{pi}, \ldots, l_n^{pn}, l_{coord}^0)$, then by rule R3, the internal advancement step $(p1_1, \ldots, pi_i, \ldots, pn_n) \xrightarrow{\tau} (p1_1, \ldots, (pi+1)_i, \ldots, pn_n)$ in L corresponds to either

- transition $(l_1^{p1}, \ldots, l_i^{pi}, \ldots, l_n^{pn}, l_{coord}^0) \xrightarrow{pch\_over_i} (l_1^{p1}, \ldots, l_i^{pi+1}, \ldots, l_n^{pn}, l_{coord}^1)$ in NTA_L, in which case I_i is the very last instance line of L to complete its prechart portion; or
- transition $(l_1^{p1}, \ldots, l_i^{pi}, \ldots, l_n^{pn}, l_{coord}^0) \xrightarrow{pch\_over_i} (l_1^{p1}, \ldots, l_i^{pi+1}, \ldots, l_n^{pn}, l_{coord}^0)$ in NTA_L, in which case I_i is not yet the last instance line of L to complete its prechart portion.

The above-mentioned first case will be followed by an intra-chart coordination, i.e., there will be an immediately following silent advancement step in L, i.e., all instance lines will move from their *Pch_bot* positions to their *Mch_top* positions simultaneously. By rule R4, the binary synchronization transition will be immediately followed by a broadcast synchronization transition $(l_1^{p1}, \ldots, l_i^{pi+1}, \ldots, l_n^{pn}, l_{coord}^1) \xrightarrow{activate}$

[3] More precisely the sub-location vector of NTA_L that is projected to $A_1||A_2|| \ldots ||A_n$. Note that the edges in *Coord* correspond only to auxiliary messages rather than the observable messages in Σ or the internal (τ) action.

[4] Note that in the LSC chart, the message sender/receiver and other relevant information are not defined as a part of the chart configuration. Accordingly, the auxiliary and bookkeeping variable information are excluded from the semantic states of the translated timed automata.

$(l_1^{p1+1}, \ldots, l_i^{pi+2}, \ldots, l_n^{pn+1}, l_{coord}^2)$, where $p1+1 = Mch\_top_{L,I_1}, \ldots, pi+2 = Mch\_top_{L,I_i}, \ldots, pn+1 = Mch\_top_{L,I_n}$. Therefore in this case, there is a correspondence between the behaviors of L and NTA_L.

When $(p1_1, \ldots, pi_i, \ldots, pn_n)$ is a position vector in the main chart of L, the unique correspondence for the case that concerns main chart completion can be proved similarly.

Now we prove the unique correspondence for the case that involves cold violations. Since an untimed chart has no conditions, a cold violation is caused only by the violation of the event partial order in the prechart. In this case, all the instance lines in the prechart of L will be brought from where they are back to their initial positions. Recall that $psn : loc(L) \to \bigcup_{I_i \in inst(L)} pos(L, I_i)$ projects a location to its position on its instance line. Formally, let us assume that L is in the cut c which corresponds to the position vector $(p1_1, \ldots, pi_i, \ldots, pj_j, \ldots,\ pn_n)$ such that $pk_k < Pch\_bot_{L,I_k}, 1 \leq k \leq n$. For any message label $m \in \Sigma$, if $\exists mo \in MO(L) \,.\, (lab(mo) = m) \wedge (\exists I_i, I_j \in inst(L) \,.\, ((src(mo) = I_i) \wedge (dest(mo) = I_j) \wedge (psn(tail(mo)) \neq pi+1) \wedge (psn(head(mo)) \neq pj+1)))$, then at cut c, the event partial order will be cold-violated by any (external) m-labeled message from I_i to I_j. For such an m-labeled message occurrence mo, by rule R6, the cold violation step $(p1_1, \ldots, pi_i, \ldots, pj_j, \ldots, pn_n) \xrightarrow{m} (0_1, \ldots, 0_i, \ldots, 0_j,\ \ldots, 0_n)$ in L uniquely corresponds to a sequence of three concatenated synchronizations in NTA_L:
$(l_1^{p1}, \ldots, l_i^{pi}, \ldots, l_j^{pj}, \ldots, l_n^{pn}, l_{coord}^0) \xrightarrow{m}$
$(l_1^{p1}, \ldots, l_i^{pi}, \ldots, l_j^{Rst}, \ldots, l_n^{pn}, l_{coord}^0) \xrightarrow{pch\_vio}$
$(l_1^{p1}, \ldots, l_i^{pi+1}, \ldots, l_j^0, \ldots, l_n^{pn}, l_{coord}^{Rst}) \xrightarrow{reset}$
$(l_1^0, \ldots, l_i^0, \ldots, l_j^0, \ldots, l_n^0, l_{coord}^0)$.

Note that according to rule R7, a hot violation in the main chart of L will end up with a semantic state that has a deadend location in a certain TA of NTA_L. This transition will *not* be considered as a part of an accepted trace of NTA_L.

In conclusion, each possible advancement step in L uniquely corresponds to a sequence of concatenated message synchronization and internal action transitions in NTA_L. They consume exactly the same message label in Σ. Therefore, each accepted trace in L uniquely corresponds to an accepted trace in NTA_L modulo the message alphabet Σ. □

Let LS be a set of untimed LSC charts $L_1, L_2, \ldots, L_n$. Each chart L_i contains the instance lines $I_{i,1}, I_{i,2}, \ldots, I_{i,in_i}$, where $1 \leq i \leq n$, and $in_i = \#(inst(L_i))$ denotes the number of instance lines in L_i. The entire translated network of TAs will be $NTA_{LS} = \{A_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq \#(inst(L_i))\} \cup \{Coord_i \mid 1 \leq i \leq n\}$. The message alphabet of LS will be the union of all the message alphabets for the individual charts, i.e., $\Pi = \bigcup_{i=1}^{n} \Sigma_i$. The alphabet of observable actions will be $Act = (\Pi \cup Aux)$.

Lemma 3 *Let LS be a set of untimed LSC charts whose message alphabet is Π, and let NTA_{LS} be the translated network of timed automata which have a set $Act = \Pi \cup Aux$ of normal and auxiliary channels. Then $\forall \gamma_1 \in (\Pi \cup \{\tau\})^{\omega} \,.\, ((\gamma_1 \models LS) \Rightarrow \exists \gamma_2 \in (Act \cup \{\tau\})^{\omega} \,.\, (\gamma_2 \models NTA_{LS}) \wedge (\gamma_2|_{\Pi} = \gamma_1|_{\Pi}))$, and $\forall \gamma_2 \in (Act \cup \{\tau\})^{\omega} \,.\, ((\gamma_2 \models NTA_{LS}) \Rightarrow \exists! \gamma_1 \in (\Pi \cup \{\tau\})^{\omega} \,.\, (\gamma_1 \models LS) \wedge (\gamma_2|_{\Pi} = \gamma_1|_{\Pi}))$.*

Proof In this case, in order to prove the above two implications, we need to prove that each *cut vector* of *LS* uniquely corresponds to a location vector in NTA_{LS}, and each advancement step in *LS* uniquely corresponds to an *equivalence class* of sequences of concatenated (broadcast) synchronization and internal action transitions in NTA_{LS}. Although elements in the equivalence class have different intermediate location vectors, they have the same initial and final location vectors. They consume exactly the same message in Π. Note that an advancement step in *LS* always represents a legal behavior.

By Lemma 2, for each untimed chart L_i in *LS*, each cut in L_i uniquely corresponds to a location vector in the corresponding network of timed automata NTA_{L_i}, and each advancement step in L_i uniquely corresponds to either a single message synchronization transition, or a sequence of concatenated message synchronization and internal action transitions in NTA_{L_i}.

The only semantic difference between the advancement steps of a single untimed chart and of a set of untimed charts is that in the latter case there exist *inter-chart* coordinations, i.e., across-chart broadcast synchronization on message occurrences of the same message is possible. This implies that:

(1) At a cut vector of *LS*, if in more than one chart there are enabled message occurrences of the same message, then either all of them are chosen to be fired simultaneously, or none of them is chosen to be fired;
(2) Due to the nature of broadcast synchronization in the translated network of TAs, while a message at a cut vector of *LS* could correspond to a legal message synchronization advancement step in a certain chart, meanwhile it could also lead another chart to be reset by cold-violating the prechart of that chart (case 2.1), or lead another chart to a deadlocked situation by hot-violating the main chart of that chart (case 2.2).

In case (1), given a set *LS* of untimed LSC charts $L_1, L_2, \dots, L_n$, we let $in_i = \#(inst(L_i))$, $1 \le i \le n$. We assume that the current cut vector $\bar{c}$ of *LS* uniquely corresponds to the position vector $(p_{1,1}, p_{1,2}, \dots, p_{1,in_1}, p_{2,1}, p_{2,2}, \dots, p_{2,in_2}, \dots, p_{n,1}, p_{n,2}, \dots, p_{n,in_n})$, where $p_{i,j} \in pos(L_i, I_j)$ denotes the current position on instance I_j of chart L_i. Without loss of generality, we assume that two *m*-labeled message occurrences mo_1 and mo_2 are enabled at cut vector $\bar{c}$ in two charts L_i and L_j, respectively. Specifically, let $(p_{i,a}+1)$ and $(p_{i,b}+1)$ be the sending and receiving positions of mo_1 in L_i, where $1 \le a, b \le in_i$, and let $(p_{j,c}+1)$ and $(p_{j,d}+1)$ be the sending and receiving positions of mo_2 in L_j, where $1 \le c, d \le in_j$. According to the trace-based semantics for a set of charts, these two message synchronization advancement steps in L_i and L_j will occur simultaneously. By rule R2, there will be an $m!$-labeled edge from location $l_{i,a}^{p_{i,a}}$ to $l_{i,a}^{p_{i,a}+1}$ in $A_{i,a}$, and an $m?$-labeled edge from location $l_{i,b}^{p_{i,b}}$ to $l_{i,b}^{p_{i,b}+1}$ in $A_{i,b}$, and similarly for chart L_j. By rule R5, an extra $m?$-labeled edge from location $l_{i,a}^{p_{i,a}}$ to $l_{i,a}^{p_{i,a}+1}$ will be added in $A_{i,a}$, and similarly in chart L_j. Consequently, there will be a broadcast synchronization on *m* among $A_{i,a}$, $A_{i,b}$, $A_{j,c}$, $A_{j,d}$, initiated either by $A_{i,a}$, or by $A_{j,c}$. In either case, after this broadcast synchronization on *m* in NTA_{LS}, the locations of $A_{i,a}$, $A_{i,b}$, $A_{j,c}$ and $A_{j,d}$ will progress to $l_{i,a}^{p_{i,a}+1}$, $l_{i,b}^{p_{i,b}+1}$, $l_{j,c}^{p_{j,c}+1}$ and $l_{j,d}^{p_{j,d}+1}$, respectively. Therefore, the message synchronization advancement step on *m* in *LS* corresponds to two possible interleaved executions among $A_{i,a}$, $A_{i,b}$, $A_{j,c}$ and $A_{j,d}$. Since both interleavings consume the same message label *m*, they correspond to the same portion of the accepted trace in NTA_{LS}. These two interleaved executions constitute an equivalence class with respect to the message synchronization advancement step on *m*.

In case (2.1), assume that the current cut vector $\bar{c}$ of *LS* corresponds to position vector $(p_{1,1}, p_{1,2}, \ldots, p_{1,in_1}, p_{2,1}, p_{2,2}, \ldots, p_{2,in_2}, \ldots, p_{n,1}, p_{n,2}, \ldots, p_{n,in_n})$. Without loss of generality, we assume that an *m*-labeled message occurrence *mo* is currently enabled in L_i, but not in L_j, and that $\bar{c}$ "cuts" the prechart of L_j. According to the semantics for a set of LSC charts, when message *m* is encountered, there will be a normal advancement step in L_i, and a cold violation advancement step in L_j. By Lemma 2, such a cold violation advancement step uniquely corresponds to a sequence of synchronizations in the relevant timed automata. Therefore, the system-wide synchronization on *m* will also uniquely correspond to a system-wide sequence of synchronizations in NTA_{LS}.

In case (2.2), assume that the current cut vector $\bar{c}$ of *LS* corresponds to position vector $(p_{1,1}, p_{1,2}, \ldots, p_{1,in_1}, p_{2,1}, p_{2,2}, \ldots, p_{2,in_2}, \ldots, p_{n,1}, p_{n,2}, \ldots, p_{n,in_n})$. Without loss of generality, we assume that an *m*-labeled message occurrence *mo* is currently enabled in L_i, but not in L_j, and that $\bar{c}$ "cuts" the main chart of L_j. According to the semantics for a set of LSC charts, when message *m* is encountered, there will be a normal message synchronization advancement step in L_i, and a hot violation in L_j. Specifically, let $p_{i,a}$ and $p_{i,b}$ be the sending and receiving positions of *mo* in L_i, where $1 \leq a, b \leq in_i$. We let the sub-position vector in L_j be $c_j = (p_{j,1}, p_{j,2}, \ldots, p_{j,in_j})$. Obviously, *mo* is not enabled at sub-cut c_j. Since L_j is hot-violated by *mo*, there must exist a position, say $p_{j,x}$, $1 \leq x \leq in_j$, such that there is an *m*?-labeled edge from $p_{j,x}$ to a sink error location `Err` in $A_{j,x}$. After label *m* is consumed, the next semantic state of NTA_{LS} will be reached. This semantic state will have a deadend location `Err`, which indicates that the system will be deadlocked. Therefore, the TA transition step leading to this semantic state will *not* be considered as a part of the accepted trace. In this case, *m* will not be allowed to occur at cut vector $\bar{c}$. This demonstrates how the different charts constrain the behaviors of each others. In summary, in case (2.2), a to-be-hot violating message in *LS* uniquely corresponds to a to-be-deadlocked TA transition in NTA_{LS}.

Based on the above discussions, we conclude that there exists a unique correspondence between the observable traces of a set of untimed LSC charts and their corresponding network of timed automata. □

Let *L* be a time-enriched chart whose instance lines $I_1, I_2, \ldots, I_n$ correspond to timed automata $A_1, A_2, \ldots, A_n$, respectively. Let the message alphabet of *L* be $\{m_1, m_2, \ldots, m_k\}$. According to Section 5.1.3 ("Translation of assignments") and Section 5.2, in the worst case there will be an auxiliary timed automaton A_{m_i} for each m_i, $1 \leq i \leq k$. Consequently, the translated network of TAs will be $NTA_L = \{A_i \mid 1 \leq i \leq n\} \cup \{Coord\} \cup \{A_{m_i} \mid 1 \leq i \leq k\}$.

According to rules R3, R4 and R6, there will be auxiliary channels $Aux = \{pch\_over_i, mch\_over_i \mid 1 \leq i \leq n\} \cup \{activate, over, pch\_vio, reset\}$ used in NTA_L. According to rule R10, there could be auxiliary channels $Aux' = \{m_i\_Rpt, m_i\_Rst, m_i\_Rcv \mid 1 \leq i \leq k\}$ used in NTA_L. Let the message alphabet of L be Σ, then the alphabet of observable actions in NTA_L will be $Act = \Sigma \cup Aux \cup Aux'$.

Lemma 4 *Let L be a time-enriched LSC chart whose message alphabet is Σ, and let NTA_L be the translated network of timed automata which have a set $Act = \Sigma \cup Aux \cup Aux'$ of normal and auxiliary channels. Then $\forall \gamma_1 \in (\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^\omega . ((\gamma_1 \models L) \Rightarrow \exists \gamma_2 \in (Act \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^\omega . (\gamma_2 \models NTA_L) \wedge (\gamma_2|_{(\Sigma \cup \mathbb{R}_{\geq 0})} = \gamma_1|_{(\Sigma \cup \mathbb{R}_{\geq 0})}))$, and $\forall \gamma_2 \in (Act \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^\omega . ((\gamma_2 \models NTA_L) \Rightarrow \exists! \gamma_1 \in (\Sigma \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^\omega . (\gamma_1 \models L) \wedge (\gamma_2|_{(\Sigma \cup \mathbb{R}_{\geq 0})} = \gamma_1|_{(\Sigma \cup \mathbb{R}_{\geq 0})}))$.*

Proof In order to prove the above two implications, we need to show that each *configuration* of chart L uniquely corresponds to a certain semantic state of NTA_L, and each advancement step in L uniquely corresponds to a sequence of concatenated message synchronization transitions, and/or internal action transitions, and/or time delay transitions in NTA_L such that they either consume exactly the same letter from Σ, or undergo exactly the same period of time delay.

By Lemma 2, each cut of an untimed chart L uniquely corresponds to a semantic state in NTA_L, and each advancement step in L uniquely corresponds to either a message synchronization transition, or a sequence of concatenated message synchronization and internal action transitions in NTA_L. For a time-enriched LSC chart, we keep this skeleton correspondence, i.e., we map position pi_i of instance line I_i to location l_i^{pi} of the timed automaton A_i. Note that along an instance line of the time-enriched chart, two adjacent LSC positions typically do not correspond to two adjacent locations in the corresponding translated TA. Between location l_i^{pi} and l_i^{pi+1}, where $0 \leq pi \leq (p\_max_{L,I_i} - 1)$, according to rules R9, R10 and R11, we will add some intermediate auxiliary TA locations, and add some TA edges to connect them.

Now we prove that a message synchronization advancement step on m in L uniquely corresponds to a sequence of transitions in NTA_L that consumes m exactly. Assume that at a configuration c which corresponds to position vector $(p1_1, \ldots, pi_i, \ldots, pj_j, \ldots, pn_n)$ in L and at clock valuation v, there is an m-labeled message occurrence mo with condition (clock constraints) g and assignment (clock resets) a sent from position $(pi+1)_i$ of instance I_i to position $(pj+1)_j$ of instance I_j. Assume that position pi_i corresponds to location l_i^{pi} in A_i, and position $(pi+1)_i$ corresponds to location l_i^{pi+1} in A_i, then there will be five intermediate locations between l_i^{pi} and l_i^{pi+1} in A_i, which we denote as $l_i^{pi,1}, l_i^{pi,2}, l_i^{pi,3}, l_i^{pi,4}$ and $l_i^{pi,5}$. Here

- between l_i^{pi} and $l_i^{pi,1}$, there is a TA edge with the guard "$m\_mayRcv ==$ `true`";
- between $l_i^{pi,1}$ and $l_i^{pi,2}$, there is a TA edge which tests the upper bound constraints;
- between $l_i^{pi,2}$ and $l_i^{pi,3}$, there is an $m!$-labeled TA edge;
- between $l_i^{pi,3}$ and $l_i^{pi,4}$, there is a TA edge which tests the lower bound and/or clock difference constraints;
- between $l_i^{pi,4}$ and $l_i^{pi,5}$, there is an $m\_Rpt!$-labeled TA edge;
- between $l_i^{pi,5}$ and l_i^{pi+1}, there is an $m\_Rst?$-labeled TA edge.

Similarly, in A_j there will also be five intermediate locations and the corresponding edges that connect them. Specifically, if there are positions on other instance lines that are waiting for the completion of this message synchronization according to the partial order relation (like the situation of Fig. 21), then there will be one more intermediate location $l_i^{pi,6}$, and an *m_Rcv*!-labeled edge connecting $l_i^{pi,6}$ to l_i^{pi+1}. According to rule R11, the position sub-vector (pi_i, pj_j) corresponds to the TA location sub-vector (l_i^{pi}, l_j^{pj}), where both locations are committed locations. After these two transitions from (l_i^{pi}, l_j^{pj}), the new location sub-vector $(l_i^{pi,1}, l_j^{pj,1})$ will be reached, which are also committed locations. Since a legal advancement step in L will not violate the upper bound of the clock constraints, the upper bound constraint will evaluate to true and thus the next location sub-vector will be $(l_i^{pi,2}, l_j^{pj,2})$. From $(l_i^{pi,2}, l_j^{pj,2})$ there will be the message synchronization on m leading to $(l_i^{pi,3}, l_j^{pj,3})$, which are again committed locations. After comparing the lower bound of clock constraints, the location sub-vector $(l_i^{pi,4}, l_j^{pj,4})$ will be reached. Now instance lines I_i and I_j will immediately report to the dedicated automaton A_m, telling it that the instances are done with testing the guarding flag boolean variables, testing the upper bound, message synchronization, and testing the lower bound or clock difference. Once both instance lines have notified A_m of their completions, A_m will immediately initiate an *m_Rst*-labeled broadcast synchronization which brings A_i from $l_i^{pi,5}$ to l_i^{pi+1}, and brings A_j from $l_j^{pj,5}$ to l_j^{pj+1}. Specifically, if there is an $l_i^{pi,6}$ in A_i, then the *m_Rst*?-labeled edge will be from $l_i^{pi,5}$ to $l_i^{pi,6}$ in A_i, and there will be an *m_Rcv*!-labeled edge from $l_i^{pi,6}$ to l_i^{pi+1}. In summary, the message synchronization step on m in L will uniquely correspond to such a sequence of transitions in NTA_L.

For a silent advancement step in L, it is the same as in the untimed case. In other words, the corresponding proof for Lemma 2 also applies here.

For a time delay advancement step in L, since the upper bounds and lower bounds of clock constraints are properly translated to tests that are prior to and after the message synchronization in NTA_L, a time delay of a period of $d \in \mathbb{R}_{\geq 0}$ is allowed in NTA_L if and only if the same period d of time delay is allowed in L.

In all the three possible cases of an advancement step in L, there will be a uniquely corresponding sequence of transitions in NTA_L such that this sequence consumes exactly the same message or the same amount of time delay as that step in L. □

Let LS be a set of time-enriched LSC charts $L_1, L_2, \ldots, L_n$. Each chart L_i contains the instance lines $I_{i,1}, I_{i,2}, \ldots, I_{i,in_i}$, where $in_i = \#(inst(L_i))$. Let the message alphabet Π of LS be $\Pi = \bigcup_{i=1}^{n} \Sigma_i = \{m_1, m_2, \ldots, m_k\}$. Then in the worst case, the translated network of TAs will be $NTA_{LS} = \{A_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq \#(inst(L_i))\} \cup \{Coord_i \mid 1 \leq i \leq n\} \cup \{A_{m_i} \mid 1 \leq i \leq k\}$. Similar to Lemma 4, we let $Act = \Pi \cup Aux \cup Aux'$ (note that here Aux and Aux' are as defined in Section 5.2).

Theorem 5 Let LS be a set of time-enriched LSC charts whose message alphabet is Π, and let NTA_{LS} be the translated network of timed automata which have a set Act of normal and auxiliary channels. Then $\forall \gamma_1 \in (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{\omega} \,.\, ((\gamma_1 \models LS) \Rightarrow \exists \gamma_2 \in (Act \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{\omega} \,.\, (\gamma_2 \models NTA_{LS}) \wedge (\gamma_2|_{(\Pi \cup \mathbb{R}_{\geq 0})} = \gamma_1|_{(\Pi \cup \mathbb{R}_{\geq 0})}))$, and $\forall \gamma_2 \in (Act \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{\omega} \,.\, ((\gamma_2 \models NTA_{LS}) \Rightarrow \exists! \gamma_1 \in (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{\omega} \,.\, (\gamma_1 \models LS) \wedge (\gamma_2|_{(\Pi \cup \mathbb{R}_{\geq 0})} = \gamma_1|_{(\Pi \cup \mathbb{R}_{\geq 0})}))$.

Proof We need to prove that each cut vector of *LS* uniquely corresponds to a location vector in NTA_{LS}, and each message synchronization advancement step in *LS* uniquely corresponds to a sequence of concatenated message synchronization transitions, and internal action transitions in NTA_{LS}. These transitions are connected by committed locations in NTA_{LS}. Because any committed location appears as a junction location only when it will be immediately followed (only) by a condition test, these concatenated transitions can be viewed as an atomic step. Although for the sake of inter-chart coordination, the outgoing transitions from locations of different TAs may be executed in an interleaved manner, the order of the consumed words in $(\Pi \cup \{\tau\})^*$ remains the same. In other words, an accepted timed trace $\gamma \in (\Pi \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^{\omega}$ may correspond to an equivalence class of timed traces in $(Act \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^*$. They consume exactly the same timed trace in $(\Pi \cup \mathbb{R}_{\geq 0})^*$. Proof details concerning the translations of inter-chart message coordinations and message occurrences that are associated with conditions and/or assignments are similar to that for Lemma 3 and Lemma 4, respectively. □

Let *LS* be an LSC system which consists of a set of (untimed or timed) driving universal charts $L_1, L_2, \ldots, L_n$. We translate *LS* to a network of timed automata NTA_{LS}. Let L' be a separate monitored universal chart (i.e., the "property chart"), which will be translated to another network of timed automata $NTA_{L'}$. Let the TA locations $Coord_{L'}.Mch\_top$ and $Coord_{L'}.Mch\_bot$ denote that the main chart of L' has just been activated and has just been successfully matched, respectively. We have:

Theorem 6 $LS \models L' \Leftrightarrow (NTA_{LS} || NTA_{L'}) \models Coord_{L'}.Mch\_top \leadsto Coord_{L'}.Mch\_bot.$

Proof By Theorem 5, each accepted trace in *LS* uniquely corresponds to a cluster of accepted traces in NTA_{LS} which consume exactly the same string from $(\Pi \cup \mathbb{R}_{\geq 0})^{\omega}$. And similarly for L' and $NTA_{L'}$.

The TA location $Coord_{L'}.Mch\_top$ represents the situation where the property chart L' has just been activated, and $Coord_{L'}.Mch\_bot$ the situation where L' has just been satisfied (i.e., successfully matched).

Since L' is a property chart, its corresponding network of timed automata $NTA_{L'}$ will never interfere with (or "drive") the network of timed automata NTA_{LS}. This means that after composing the TAs in $NTA_{L'}$ with the TAs in NTA_{LS}, the behaviors in NTA_{LS} will not be further constrained. Since both $Coord_{L'}.Mch\_top$ and $Coord_{L'}.Mch\_bot$ are locations in the product automaton of $(NTA_{LS} || NTA_{L'})$, the right hand side formula of this theorem captures exactly the assume-guarantee style responsiveness property of the LSC requirement, which is exactly what we require of $LS \models L'$. □

An LSC system *LS* satisfies a monitored existential chart L' iff one of the traces in *LS* is included in the traces of L'.

Theorem 7 $LS \models L' \Leftrightarrow (NTA_{LS} || NTA_{L'}) \models \mathsf{E}\Diamond\ Coord_{L'}.Mch\_bot.$

Proof (idea) This theorem can be proved similarly as in Theorem 6, except that an existential chart has no prechart and one satisfying run suffices to prove this property. □