Verifying real-time systems against scenario-based requirements

Larsen, Kim Guldstrand; Li, Shuhao; Nielsen, Brian; Pusinskas, Saulius

Published in:
Lecture Notes in Computer Science

DOI (link to publication from Publisher):
10.1007/978-3-642-05089-3_43

Publication date:
2009

Document Version
Publisher's PDF, also known as Version of record

Verifying Real-Time Systems against Scenario-Based Requirements

Kim G. Larsen, Shuhao Li, Brian Nielsen, and Saulius Pusinskas

Center for Embedded Software Systems (CISS)
Aalborg University
Selma Lagerlöfs Vej 300, DK-9220 Aalborg, Denmark
{kgl, li, bnielsen, saulius}@cs.aau.dk

Abstract. We propose an approach to automatic verification of real-time systems against scenario-based requirements. A real-time system is modeled as a network of Timed Automata (TA), and a scenario-based requirement is specified as a Live Sequence Chart (LSC). We define a trace-based semantics for a kernel subset of the LSC language. By equivalently translating an LSC chart into an observer TA and then non-intrusively composing this observer with the original system model, the problem of verifying a real-time system against a scenario-based requirement reduces to a classical real-time model checking problem. We show how this is accomplished in the context of the Uppaal model checker.

1 Introduction

A model checker typically needs two inputs: a model $\mathcal{M}$ characterizing the state-transition behaviors of a finite state concurrent system, and a temporal logic formula $\mathcal{P}$ specifying the properties of interest. For real-time model checkers such as KRONOS [20] and Uppaal [3], $\mathcal{M}$ might be a network of Timed Automata (TA) [1], and $\mathcal{P}$ might be a formula of the TCTL logic [20] or a fragment of the CTL logic [3]. While the enhanced versions of TA are relatively expressive modeling formalisms, the TCTL or CTL logics appear to be property specification languages of only limited capability, intuitiveness, and convenience:

- The atomic propositions can only be state propositions, where messages (events) are not allowed to appear [20, 3];
- There is no means for specifying non-trivial quantitative timing constraints (e.g., there is no time-bounded temporal operator like $\Diamond_{[1,3]}$) [3].

These limitations imply that straightforward characterizations of event synchronizations, causal relations, or 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" as a query in KRONOS and Uppaal are not possible.

Essentially, the query languages of these model checkers describe only *intra*-process 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 parallelly composed system model) satisfy some particular properties. In contrast, the *inter*-process properties describe how different processes interact, collaborate and cooperate via message or rendezvous synchronizations.

Live Sequence Chart (LSC) [11] is a visual formalism for scenario-based specification and programming. It extends the classical Message Sequence Charts (MSC) [13] by adding modalities[1]. The LSC language is unambiguous because it has strictly defined semantics, e.g., the executable (operational) semantics [11] and the trace-based semantics [7].

We envisage LSC as a nice complement to the intra-process property specification language of (real-time) model checkers in general and of Uppaal in particular:

- *Intuitiveness.* LSC has the necessary language constructs to describe a variety of causality and non-trivial scenarios. As a visual formalism, LSC is more intuitive in capturing scenario-based user requirements than the CTL fragment of Uppaal in its textual form;
- *Expressiveness.* It has been shown that a kernel subset [15] of LSC can be embedded into CTL^*, *provided that* event occurrences can be used as atomic propositions [15]. This indicates that LSC cannot always be encoded as CTL^*;
- *Counterexample display.* LSC provides the possibility of displaying counterexamples also in the requirement specifications.

In this paper we capture a scenario that is to be verified using an LSC chart. We obtain a behavior-equivalent observer TA from this chart by mapping the LSC cuts and discrete advancement steps to TA locations and edges, respectively. We let the observer TA spy on the relevant events of the original system via model instrumentation, semaphore locking, and parallel composition. In this way, the problem of verifying a real-time system against a scenario-based requirement will be reduced to a classical model checking problem in Uppaal.

1.1 Related work

To model check real-time systems against complex properties or scenario-based requirements, various approaches have been proposed.

One solution is the observer automata approach [4], i.e., to construct a number of auxiliary TA to capture the scenario-based requirements, and then parallelly compose them with the original TA models. While this method can be practically useful [16], there are some limitations: (1) manual constructions of observer TA could be labor-intensive and error-prone. To be composed with the observer TA, the original system model may need to be modified; (2) the observer TA and the original system engage in normal channel synchronizations, thus specifying process interactions only *liberally* (i.e., no particular sending and

[1] The *existential* and *cold* (resp. *universal* and *hot*) modalities represent the provisional (resp. mandatory) requirements. For example, an existential (resp. universal) chart specifies restrictions over at least one satisfying (resp. all possible) system runs; a cold condition may be violated, whereas a hot one must be satisfied.

receiving process is specified for a message). In our verification framework, automatic construction of observers from LSC charts overcomes both problems.

Another line of research is first to capture the scenario-based requirements using the assume-guarantee style visual formalisms such as Triggered MSC [19], Template MSCs [10], or the even richer LSC [11], and then transform them into directly verifiable formalisms. In particular LSCs can be translated into Timed Büchi Automaton (TBA) [14], TA [17], temporal logic [14, 12, 15, 8, 6], or sequences of LSC elements [18], and the verification problem can be converted to a classical model checking problem [14], or solved directly [18].

In [14] an LSC chart is transformed into a TBA. To specify real-time requirements, timers [2, 13] and timing annotations (or delayed intervals) [2] 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 system model, and enables smooth translation of timing information into the observer TA, and seamless embedding of the observer TA into the Uppaal verification framework.

An LSC-to-TA translation has been proposed in [17], which inspires our current translation. Since we use LSC only as a property specification language, and not also as a modeling language [17], we define a clearer semantics, according to which there is no need to translate one LSC chart into multiple TA as in [17].

LSCs can also be translated into temporal logic formulas [12, 15, 8, 6]. For the kernel subset of LSC in [15], it has been shown that existential charts can be expressed using the CTL logic, and universal charts can be expressed using (LTL ∩ CTL) [12, 15]. Similar results are achieved in [8]. However, these methods do not handle explicit time in the charts, and the extraordinary size of the resulting formula limits the scale of the charts that can be translated and verified.

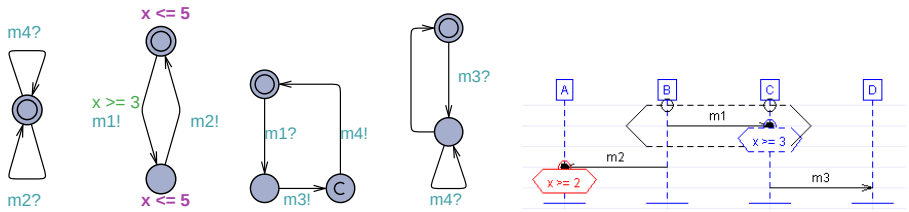
In [18] properties are extracted from LSCs as sequences of LSC elements, and algorithms have been developed to check whether these sequences are respected by the FSM computation graph of the TA model that is exported from Uppaal. However, simultaneous regions (simregions) in LSCs are used only to model broadcast communications, and conditions cannot be a part of simregions. Our notion of simregion uses the "[condition] [message]/[assignment]" pattern, thus enables smooth translation into a TA edge.

1.2 Contributions

The contributions of this paper include: (1) we define a kernel subset of the LSC language that is suitable for capturing scenario-based requirements of real-time systems, and define a trace-based semantics; (2) we propose a behavior-equivalent translation of an LSC chart into a TA; (3) we present a method of embedding the translated TA into Uppaal, thus encoding the problem of verifying systems against LSC requirements as a classical model checking problem.

2 Modeling and specification of real-time systems

In Uppaal, a real-time system is modeled as a network of TA. These TA communicate on shared global variables, or via handshaking on CCS-style synchronization channels. Standard constructs of TA include locations, edges, clock constraints, clock resets, and location invariants. In addition, Uppaal has a number of extensions [3] to the TA formalism such as urgent and committed locations[2], broadcast channels, data variables, variable constraints and updates, etc. Fig. 1(a)-1(d) give an example of a network of TA.



(a) TA A (b) TA B (c) TA C (d) TA D (e) LSC chart $\mathcal{L}$

Fig. 1. A real-time system model (network of TA) and its requirement (LSC chart).

Uppaal uses a fragment of the CTL logic as its property specification language. Atomic propositions can only be state propositions. 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 ($\mathsf{A}\Diamond\phi$, $\phi \rightsquigarrow \varphi$). In particular the *leads-to* or *response* property $\phi \rightsquigarrow \varphi$ is a shorthand for $\mathsf{A}\Box(\phi \Rightarrow \mathsf{A}\Diamond\varphi)$, meaning that whenever ϕ is satisfied, then eventually φ will be satisfied.

Although a lot of properties can be specified using these patterns, many others still cannot. Consider a user requirement on the TA in Fig. 1: **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). Clearly, this scenario cannot be specified as a Uppaal CTL formula.

However, the above scenario requirement can be easily captured using Live Sequence Charts (Fig. 1(e)). For instance, the first block of diagrammatic elements $\{m_1, x \geq 3\}$ means that: when m_1 in the original model is observed, the clock value of x should be no less than 3 at that time. If this is the case, then the monitored execution continues; otherwise, it is a cold violation of the prechart[3].

[2] A *committed* location is a TA location where time is frozen, and the outgoing transitions have higher priority to be taken than those from non-committed ones.

[3] A universal chart can optionally contain a *prechart*, which specifies the scenario which, if successfully executed, forces the system to satisfy the scenario given in the actual chart body (the *main chart*).

3 From LSC to Uppaal Timed Automata

3.1 Live Sequence Chart

We consider the following LSC elements: instance, location, message, clock variable, condition, assignment, and simregion.

An LSC chart can have a role, a type, and an activation mode. In this paper we consider the role of *system 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 or reset those clocks. We consider the *universal* type charts. Furthermore, we consider the *invariant* activation mode, i.e., the chart will be activated whenever a minimal event (i.e., an event that is minimal in the partial order induced by the chart) is matched, regardless of the state of the main chart.

Each LSC chart $\mathcal{L}$ describes a particular interaction scenario of a set I of processes (or instances, or agents). Along each instance line $i \in I$ there are a finite set of "positions" $pos(i) = \{0, 1, \ldots, p\_max_i\}$, which denote the possible points of communication and computation. We denote all *locations* of $\mathcal{L}$ as $L = \{\langle i, p\rangle | i \in I \ \wedge \ p \in pos(i)\}$.

Let Σ be the alphabet of messages (a.k.a. "channels" in Uppaal). A *message* $m = (\langle i, p\rangle, \sigma, \langle i', p'\rangle) \in L \times \Sigma \times L$ corresponds to instance i, while in position p, sending σ to instance i' at position p'. The (finite) set of all messages appearing in $\mathcal{L}$ are denoted as M. We call σ the message *label*. We say that i and i' are the source (src) and destination (dest) instances, respectively. Messages are assumed to be instantaneous (thus we use the terms *message* and *event* interchangeably). Furthermore, messages are assumed to be of *hot* temperature, i.e., they never get lost during transmission. This paper does not consider concurrent messages, thus each location can be the end point of at most one message in the chart.

Let the finite sets of real-valued *clock variables* (ranging over $\mathbb{R}_{\geq 0}$) of $\mathcal{L}$ and of the original system model $\mathcal{S}$ be $C_{\mathcal{L}}$ and $C_{\mathcal{S}}$, respectively. The set of readable clock variables in $\mathcal{L}$ will be $C = C_{\mathcal{L}} \cup C_{\mathcal{S}}$. Since $\mathcal{L}$ is a monitored chart, only clocks in $C_{\mathcal{L}}$ can be reset in the chart.

A *clock constraint* is of the form $x \bowtie n$ or $x - y \bowtie n$ where $x, y \in C$, $n \in \mathbb{Z}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$. A *condition* is a finite conjunction of clock constraints. The set of conditions are denoted G. Conditions may be either hot or cold.

A *clock reset* is of the form $x := 0$ where $x \in C_{\mathcal{L}}$. An *assignment* a is a finite set of clock resets. For simplicity it is denoted as a set a of clocks to be reset. The set of all assignments is $A = 2^{C_{\mathcal{L}}}$.

When there is a message m sent from one instance i_1 to another instance i_2, the message anchoring point on i_1 or i_2 could be associated with a condition g and/or an assignment a. The condition g is a predicate which is evaluated immediately *after* the message has been observed, and the assignment is a reset of the clocks in a providing that g evaluates to true. The message, condition and assignment can be collectively viewed as an atomic step of LSC execution, i.e., they take place at the same time, hence the notion of *simultaneous region* (simregion), which is inspired by [14].

Definition 1 (simregion). *A* simregion s *is a set of LSC message, condition, and assignment,* $s \subseteq (M \cup G \cup A)$*, which satisfy the following requirements:*

- *non-emptiness:* $\exists e \in (M \cup G \cup A).e \in s$*;*
- *uniqueness:* $\forall m, n \in M.(m \in s \wedge n \in s) \Rightarrow m = n$*; (similarly for condition and assignment.)*
- *non-overlapping: for any two simregions* s *and* s'*, we have* $\forall e \in (M \cup G \cup A).(e \in s \wedge e \in s') \Rightarrow s = s'$*.* □

We write a simregion as $s = \{m, g, a\}$, where m, g, and a represent the message, condition, and assignment, respectively. The set of all simregions is denoted $S \subseteq 2^{(M \cup G \cup A)}$.

A message spans across two instance lines. A condition spans across one or more instance lines. In a simregion, the message, condition and assignment (if any) have exactly one common anchoring point. If a simregion s has a message, then the condition and/or assignment (if any) of s anchor either at the message head, or at the message tail. If a simregion s has no message, then s consists of a condition test, or an assignment, or both of them combined and anchored together (possibly across multiple instance lines). In this case, s is called a *non-message* simregion. 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 previous simregion.

Fig. 1(e) is an example LSC chart, where there are three simregions $s_1 = \{m_1, x \geq 3\}$, $s_2 = \{m_2, x \geq 2\}$, and $s_3 = \{m_3\}$.

3.2 Trace-based semantics

We define $\lambda : L \rightarrow S \cup \{nil\}$ as a labeling function. For location $l \in L$, if $\lambda(l) \in S$, then there is a simregion anchoring at l; if $\lambda(l) = nil$, then l represents an entry/exit point of the prechart(Pch)/main chart(Mch).

Locations of an LSC chart are partially ordered. The partial order relation $\preccurlyeq \subseteq L \times L$ is determined by the following rules:

- Along each instance line, if location l_1 is above l_2, then $l_1 \preccurlyeq l_2$;
- All locations in the same simregion have the same order, $\forall s \in S, \forall l, l' \in L.(\lambda(l) = s) \wedge (\lambda(l') = s) \Rightarrow (l \preccurlyeq l') \wedge (l' \preccurlyeq l)$.

Definition 2 (cut). *A* cut *is a downward-closed set of locations that span across all the instance lines. Downward-closure means that if a location* l *is included in cut* c*, so are all of its preordered locations:* $\forall c \subseteq L, \forall l, l' \in L.(l \in c \wedge l' \preccurlyeq l) \Rightarrow l' \in c$*.* □

We define $loc : (S \cup 2^L) \rightarrow 2^L$ to map a simregion $s \in S$ to a set $loc(s)$ of locations that it anchors, and to map a cut $c \in 2^L$ to its *frontier* $loc(c)$, which is a set of locations that constitute the downward border line progressed so far.

Let $c \subseteq L$ be a cut, and $s \in S$ be a simregion that follows c immediately. A cut c' is an *s-successor* of c, denoted $c \xrightarrow{s} c'$, if c' is achieved by adding the set of

locations that s anchors into c, or formally, $c \xrightarrow{s} c' \Leftrightarrow \forall l \in loc(s).(l \notin c) \wedge (c' = c \cup loc(s))$.

A cut c is *minimal* (denoted $\top$) if each location in c is a top location of some instance line, and c is *maximal* (denoted $\bot$) if the bottom locations of all instance lines are included in c. A minimal or maximal cut represents a compulsory synchronization for all involved instances. Thus the partial order relation $\preccurlyeq$ on L is extended as follows:

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

Specifically, we view the maximal cut of the prechart and the minimal cut of the main chart as the same cut, i.e., $Pch.\bot = Mch.\top$.

If cut c has $c' = Mch.\bot$ as its s-successor, then we override c' as $Pch.\top$ (if any) or $Mch.\top$, which represents the situation where a universal chart goes back to its initial state upon the successful completion of a round of monitoring.

We assume that all cuts have the hot temperature.

For instance in Fig. 1(e), 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 been stepped over. Clearly, cuts $\{s_1, s_2\}$ and $\{s_1, s_3\}$ are the s_2-successor and s_3-successor of cut $\{s_1\}$, respectively.

Definition 3 (configuration). *A* configuration *of an LSC chart is a tuple (c, v), where c is a cut, and v maps each clock variable to a non-negative real number, $v : C_{\mathcal{L}} \rightarrow \mathbb{R}_{\geq 0}$.* □

For $d > 0$, notation $(v + d) : C_{\mathcal{L}} \rightarrow \mathbb{R}_{\geq 0}$ means that the function v is shifted by d such that $\forall x \in C_{\mathcal{L}}.v(x + d) = v(x) + d$.

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

An assignment $a \in A$ can be viewed as a transformer for function v, thus $a(v)$ represents the new valuation after the assignment.

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

There are three kinds of valid advancement steps between two configurations:

- *Synchronization step.* Given a chart configuration (c, v), and a simregion s which has a message m, and optionally a condition g, and/or an assignment a. There is a synchronization step $(c, v) \xrightarrow{m} (c', a(v))$ if, $c \xrightarrow{s} c'$ and $v \models g$;
- *Silent step.* Given a chart configuration (c, v), and a simregion s which optionally has a message m, and/or a condition g, and/or an assignment a. There is a silent step $(c, v) \xrightarrow{\tau} (c', a(v))$ if either
  - (*silent advancement*). $(\nexists m \in M.m \in s)$, and $v \models g$, and $c \xrightarrow{s} c'$; or
  - (*premature termination*). $g.\text{temp} = \text{cold}$, and $v \not\models g$, and $c' = Pch.\top$;

– *Time delay step.* Given a chart configuration (c, v). There is a time delay step $(c, v) \xrightarrow{d} (c, v + d)$ if there exists a simregion that follows cut c, and the clock constraints in its conditions (if any) will be satisfied after delay d, i.e., $\exists s = \{m, g, a\}.(v + d) \models g$.

Definition 4 (run). *A* run *of a universal LSC chart is a sequence of configurations* $(c, v)^0 \cdot (c, v)^1 \cdot \ldots$ *that are connected by the advancement steps, i.e.,* $\forall i \geq 0. \exists u_i \in (M \cup \{\tau\} \cup \mathbb{R}_{\geq 0}).(c, v)^i \xrightarrow{u_i} (c, v)^{i+1}$. □

The transition relation $\rightarrow$ as mentioned above each time consumes only a single letter $u \in (M \cup \{\tau\} \cup \mathbb{R}_{\geq 0})$. We extend it to $\rightarrow^*$ such that it consumes a (finite or infinite) word $w \in (M \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^* \cup (M \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^\omega$.

Let Π be the alphabet of all possible advancement steps in the original system model, which subsumes $(M \cup \{\tau\} \cup \mathbb{R}_{\geq 0})$ and can in addition include other messages not ever appeared in M.

Definition 5 (satisfaction of a prechart/main chart). *A finite timed trace* $\gamma \in \Pi^*$ satisfies *an LSC prechart or main chart* $\mathcal{C}$ *if its projection* $\gamma|_{(M \cup \{\tau\} \cup \mathbb{R}_{\geq 0})}$ *has a prefix* μ *which is the accepted word of a run that starts from the initial configuration and ends in a maximal cut configuration of* $\mathcal{C}$*, i.e.,* $\gamma \models \mathcal{C} \Leftrightarrow \exists \mu, \xi \in (M \cup \{\tau\} \cup \mathbb{R}_{\geq 0})^*.(\gamma|_{(M \cup \{\tau\} \cup \mathbb{R}_{\geq 0})} = \mu \cdot \xi) \wedge \exists v'.(\top, v_0) \xrightarrow{\mu}{}^* (\bot, v')$. □

If a universal chart $\mathcal{L}$ has no prechart Pch, then it is treated as being satisfied by an empty word.

We define $\Vdash$ to denote that a finite trace $\gamma \in \Pi^*$ satisfies chart $\mathcal{C}$ *exactly*: $\gamma \Vdash \mathcal{C} \Leftrightarrow (\gamma \models \mathcal{C}) \wedge \forall \alpha, \mu, \beta \in \Pi^*.(\alpha \cdot \mu \cdot \beta = \gamma) \wedge (\alpha \neq \varepsilon \ \vee \ \beta \neq \varepsilon) \Rightarrow (\mu \not\models \mathcal{C})$.

We define $\Vdash\!\!\!\Vdash$ to denote that chart $\mathcal{C}$ is satisfied by the *prefix* of a trace $\gamma \in \Pi^* \cup \Pi^\omega$: $\gamma \Vdash\!\!\!\Vdash \mathcal{C} \Leftrightarrow \exists \alpha \in \Pi^*, \beta \in \Pi^* \cup \Pi^\omega.(\alpha \cdot \beta = \gamma) \wedge \alpha \Vdash \mathcal{C}$.

Now we define the satisfaction relation for a full universal chart:

Definition 6 (satisfaction of universal LSC chart). *A timed trace* $\gamma \in \Pi^\omega$ satisfies *a universal chart* $\mathcal{L}$ *iff, whenever a subtrace of* γ *matches the prechart, then the main chart is matched immediately afterwards,* $\gamma \models \mathcal{L} \Leftrightarrow \forall \alpha, \mu \in \Pi^*, \beta \in \Pi^\omega.(\alpha \cdot \mu \cdot \beta = \gamma) \wedge (\mu|_{(M \cup \{\tau\} \cup \mathbb{R}_{\geq 0})} \Vdash Pch) \Rightarrow \beta|_{(M \cup \{\tau\} \cup \mathbb{R}_{\geq 0})} \Vdash\!\!\!\Vdash Mch$. □

A timed language $Lang \subseteq \Pi^\omega$ satisfies $\mathcal{L}$, denoted $Lang \models \mathcal{L}$, iff, $\forall \gamma \in Lang.\gamma \models \mathcal{L}$. Clearly, $Lang$ characterizes the system behaviors that respect $\mathcal{L}$.

For a network $\mathcal{S}$ of timed automata, we use $\mathcal{S} \models \mathcal{L}$ to denote that the timed traces (language) of $\mathcal{S}$ satisfy LSC $\mathcal{L}$.

3.3 LSC to TA translation

For each LSC chart $\mathcal{L}$, we construct a Uppaal TA $O_{\mathcal{L}}$. The basic idea is that for each cut of the LSC, we assign a TA location in Uppaal; for each discrete advancement step (i.e., a simregion) that connects two consecutive cuts, we assign a TA edge. The translation is conducted incrementally based on the partial order relation $\preccurlyeq$.

3.3.1 Determining the partial order on LSC simregions

By analyzing the graphical layout of the LSC chart, the partial order $\preccurlyeq$ on the set L of locations is determined according to the rules given in Section 3.2.

Since an advancement of a cut is caused by stepping over a simregion, the partial order $\preccurlyeq$ on L can thus be lifted to $\preccurlyeq'$ on $S \cup \{Pch.\top, Mch.\top, Mch.\bot\}$ as follows: $\forall s_1, s_2 \in (S \cup \{Pch.\top, Mch.\top, Mch.\bot\}).(s_1 \preccurlyeq' s_2 \Leftrightarrow \exists l_1 \in loc(s_1), l_2 \in loc(s_2).l_1 \preccurlyeq l_2)$.

For instance in Fig. 1(e), 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 $s_1 \preccurlyeq' s_3$.

3.3.2 Translating LSC cut into TA location

The initial cut of an LSC chart is the minimal cut $\top$ of the prechart (if any) or of the main chart (otherwise). While respecting $\preccurlyeq'$, the cut advances towards $Mch.\bot$ by stepping over simregions. Each time a simregion is stepped over, a new cut is reached.

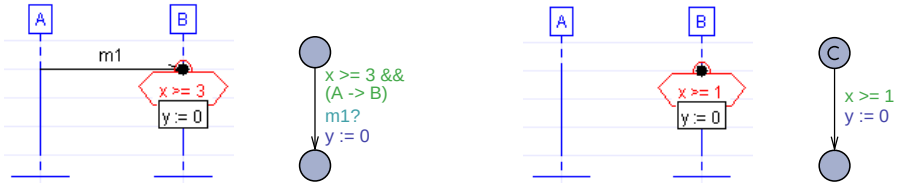
If we view all the instances of an LSC chart collectively as a whole system, then a cut can be viewed as a "location" of the TA of this whole system. For the minimal cut of the prechart (if any) and the minimal and maximal cuts of the main chart, we assign the TA locations l_{pmin}, l_{min}, and l_{max}, respectively. Note that l_{max} is a committed location, which will be connected to l_{pmin} (if any) or l_{min} via an edge of internal action transition, meaning that a next round of monitoring will begin immediately. The l_{pmin}, l_{min}, and l_{max} locations are three mandatory synchronization points for all the instances in the chart.

Time can elapse while staying in an LSC cut just like in a TA location. Specifically, a cut that is followed by a non-message simregion corresponds to a committed TA location. In that cut time is frozen and cannot elapse.

Since there are only finitely many instances and finitely many simregions in an LSC chart, the number of cuts will also be finitely many.

3.3.3 Translating LSC simregion into TA edge

If s is a message-simregion, then we map the message, condition (if any) and assignment (if any) of s into one edge of the TA. See Fig. 2(a)-2(b).



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

Fig. 2. From simregion to TA edge.

Due to the restriction of Uppaal that broadcast channels cannot be guarded by timing constraints, in the TA of Fig. 2(b), m_1 cannot be simply treated as a broadcast channel. Instead, some spying techniques will be adopted such that the translated TA will be notified of each message synchronization in the original system *immediately after* its occurrence (cf. Section 4.1).

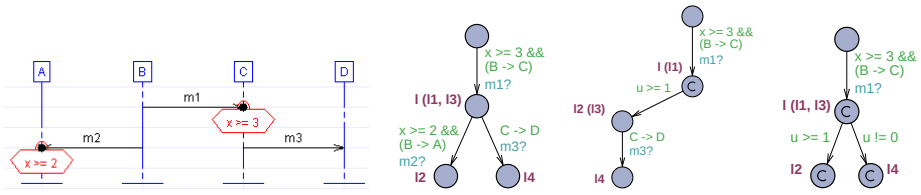
In an LSC chart, a message m is sent from one particular instance to another one (e.g., from A to B). To preserve this sender/receiver information in the translated TA, the TA edge will be further guarded by the predicate $A \rightarrow B$ (shorthand for "$src = A$ && $dest = B$"). See Fig. 2(b).

If s is a non-message simregion, then the ASAP semantics is adopted. To enforce the ASAP semantics, the source location of the translated TA edge will be marked as a committed location. See Fig. 2(c)-2(d) for an example.

3.3.4 Incremental construction of the TA

The LSC to TA translation is carried out incrementally. Assume that a TA location l has already been created for the current LSC cut (see Fig. 3(b), location l, and Fig. 3(a), cut $\{s_1\}$). Following that cut there could be a number of simregions that can be stepped over. Each of them should correspond to an outgoing edge from TA location l. Without loss of generality, we assume that there are two such immediately following simregions s_2 and s_3.

If s_2 and s_3 are both message-simregions (Fig. 3(a)), then the two new TA edges will be concatenated to location l. 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. See Fig. 3(b).



(a) The simregions (b) case #1 (c) case #2 (d) case #3

Fig. 3. TA edge construction for two subsequent simregions.

If in Fig. 3(a) s_2 is replaced by a non-message simregion, then according to the ASAP semantics, the edge (l_1, l_2) will be executed immediately, and edge (l_3, l_4) will follow, but cannot be the other way around. When concatenating these two edges to the TA, we mark l_1 as a committed location, and superpose it on l. There is only one possible interleaving where edge (l_3, l_4) follows (l_1, l_2). See Fig. 3(c).

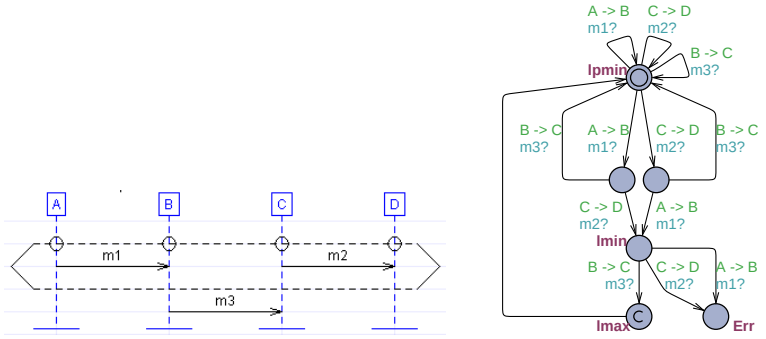
If in Fig. 3(a) s_2 and s_3 are both non-message simregions, then according to the ASAP semantics, both (l_1, l_2) and (l_3, l_4) will be executed immediately, therefore the executions will be interleaved. See Fig. 3(d).

3.3.5 Implicitly allowed behavior

In addition to the *explicitly* specified behaviors in the chart, there are also *implicitly* allowed behaviors that are due to: (1) unconstrained events, (2) cold violations, and (3) prechart pre-matching.

Let $Chan$ be the set of channels of $\mathcal{S}$, and $Chan' \subseteq Chan$ be the set of channels of $\mathcal{L}$. Clearly, channels in $(Chan \backslash Chan')$ are not constrained by $\mathcal{L}$. For each message m whose label belongs to $(Chan \backslash Chan')$, we add an m?-labeled self-loop edge to each non-committed location l of the translated TA $O_{\mathcal{L}}$. For readability they are not shown in Fig. 4.

According to the LSC semantics, cold violations of prechart or main chart are not failures. Instead, they just bring the chart back to the 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} (if Pch exists) or l_{min} (otherwise) to correspond to the $\neg g$ conditions (of DNF form). Similarly, given a cut c in the prechart, for each message m that occurs in $\mathcal{L}$ but does not follow c immediately, we also add an m?-labeled edge (l, l_{pmin}). See Fig. 4.



(a) An LSC chart (b) The translated TA

Fig. 4. Prechart matching.

According to the LSC semantics, under invariant mode the prechart will be continuously monitored. Thus for instance in Fig. 4(a), the sequence $m_1 \cdot m_1 \cdot m_2$ will match the prechart. To enforce this semantics, for each message m that appear in the chart, we add an m?-labeled self loop to location l_{pmin}.

3.3.6 Undesired behavior

The construction of the 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 m is not enabled at cut c, 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 violation we add an edge to Err.

3.3.7 Complexity

Let the number of simregions appearing in $\mathcal{L}$ be n. In the worst case, the number of locations in the translated TA $O_{\mathcal{L}}$ is $2^n + 1$. This happens when $\mathcal{L}$ consists of only the prechart or the main chart, and the messages in $\mathcal{L}$ are totally unordered.

The number of outgoing edges from a location l of $O_{\mathcal{L}}$ depends on: (1) the number of unconstrained events, ue; (2) the number of the following simregions in the corresponding cut c of $\mathcal{L}$, 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 is at the level $O(ue + fs + lc + cv)$.

3.4 Equivalence of LSC and TA

Since all the clocks in the original system model $\mathcal{S}$ are also visible to the LSC chart $\mathcal{L}$, we extend the configuration of $\mathcal{L}$ from $C_{\mathcal{L}}$ to $C_{\mathcal{L}} \cup C_{\mathcal{S}}$.

If in the translated $O_{\mathcal{L}}$ 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:

Lemma 1. *If a configuration (c, v) of $\mathcal{L}$ corresponds to a semantic state (l, v) of $O_{\mathcal{L}}$, then: (1) each simregion s that follows (c, v) in $\mathcal{L}$ uniquely corresponds to an outgoing edge (l, l') in $O_{\mathcal{L}}$, and (2) the target configuration (c', v') of s in $\mathcal{L}$ uniquely corresponds to the target semantic state (l', v') in $O_{\mathcal{L}}$.* □

Theorem 1. *For any trace tr in $O_{\mathcal{L}}$: $tr \models \mathcal{L} \Leftrightarrow (O_{\mathcal{L}}, tr) \models (l_{min} \rightsquigarrow l_{max})$.* □

Proofs of the lemmas and theorems can be found at the authors' webpages.

The prechart pre-matching mechanism does introduce undesired extra behaviors and non-determinacy. For instance in Fig. 4(b), $tr = m_1 \cdot m_2 \cdot m_1 \cdot m_2 \cdot m_3$ could be an accepted trace in $O_{\mathcal{L}}$. But since its prefix $tr' = m_1 \cdot m_2 \cdot m_1$ can be rejected, thus tr does not really satisfy $\mathcal{L}$. It coincides that the particular trace tr in the model $O_{\mathcal{L}}$ does not satisfy the property $(l_{min} \rightsquigarrow l_{max})$.

Theorem 1 indicates that $O_{\mathcal{L}}$ has exactly the same set of *legal* traces as $\mathcal{L}$.

4 Embedding into Uppaal

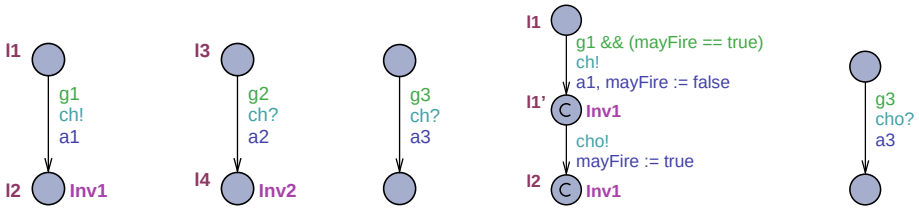
4.1 Synchronizing with the original system

When composing $O_{\mathcal{L}}$ with $\mathcal{S}$, we want $O_{\mathcal{L}}$ to "observe" $\mathcal{S}$ in a *timely* and *non-intrusive* manner. To this end, for each channel $ch \in Chan$, we make the following modifications:

(1) In $\mathcal{S}$ (e.g., Fig. 5(a)-5(b)), for each edge (l_1, l_2) that is labeled with ch!, we add a 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 $O_{\mathcal{L}}$ of the occurrence of the ch-synchronization in $\mathcal{S}$. The location invariant (if any) of l_2 will be copied on to l_1'. Furthermore, we use a global

boolean flag variable (or 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 $\mathcal{S}$ is taken, and it is set again immediately after the cho-synchronization is taken. See Fig. 5(d).

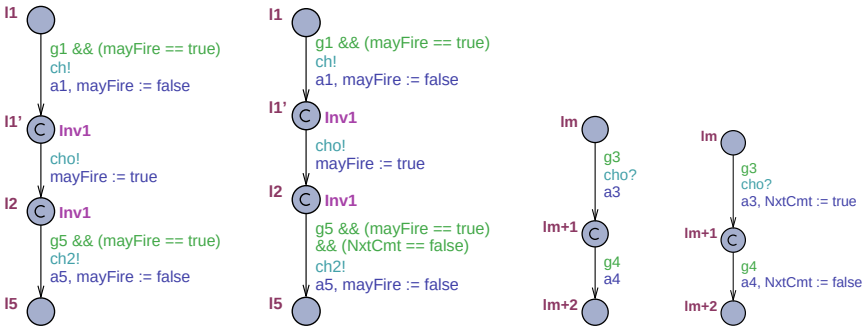
(2) In $O_{\mathcal{L}}$, each synchronization label ch? is renamed to cho?. See Fig. 5(c), 5(e).



(a) emt. edge (b) recv. edge (c) obs. edge (d) mod. emt. edge (e) mod. obs. edge

Fig. 5. Edge modifications in the original system model $\mathcal{S}$ and the observer TA $O_{\mathcal{L}}$.

If $\mathcal{L}$ has non-message simregions, then $O_{\mathcal{L}}$ has committed locations. If in a certain state both $O_{\mathcal{L}}$ and some TA in $\mathcal{S}$ are in committed locations (e.g., l_{m+1} in Fig. 6(c), l_2 in Fig. 6(a)), there will be a racing condition. But according to the ASAP semantics of $\mathcal{L}$, the (internal action) edge in $O_{\mathcal{L}}$ has higher priority. To this end, for each edge (l_i, l_{i+1}) in $O_{\mathcal{L}}$, 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 (or semaphore) $NxtCmt$ denotes whether the observer TA will be in a committed location. This semaphore is initialized to false at system start. See Fig. 6(d). Accordingly, for each ch-labeled edge (l_i, l_{i+1}) in $\mathcal{S}$ where $ch \in Chan$ and l_i is a committed location, we add "$NxtCmt$ == false" to the condition of the edge, see Fig. 6(b).



(a) emitting edge (b) modified emitting edge (c) in obs. TA (d) in modified obs. TA

Fig. 6. Edge modifications when there are committed locations in the obs. TA.

Our method of composing the observer TA $O_{\mathcal{L}}$ with the original model $\mathcal{S}$ is similar to that of [9]. 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 $\mathcal{S}$ 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 $O_{\mathcal{L}}$.

Since our method involves only syntactic scanning and manipulations, the method is not expensive. For each $ch \in Chan$, 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 $\mathcal{S}$. Moreover, we need two global boolean flag variables (*mayFire*, *NxtCmt*) as the semaphores.

4.2 Verification problem

After the modifications, the original system model $\mathcal{S}$ becomes $\mathcal{S}'$, and the observer TA $O_{\mathcal{L}}$ for chart $\mathcal{L}$ becomes $O'_{\mathcal{L}}$. Let the minimal and maximal cuts of the main chart of $\mathcal{L}$ correspond to locations l_{min} and l_{max} of $O'_{\mathcal{L}}$, respectively. Recall that the Uppaal "leads-to" property ($\phi \rightsquigarrow \varphi$) stands for $\mathsf{A}\Box(\phi \Rightarrow \mathsf{A}\Diamond\, \varphi)$, where ϕ, φ are state formulas.

Lemma 2. *If $O_{\mathcal{L}}$ has no committed location, and all $ch \in Chan$ are binary synchronization channels, then $\mathcal{S} \models \mathcal{L} \Leftrightarrow (\mathcal{S}'||O'_{\mathcal{L}}) \models (l_{min} \rightsquigarrow l_{max})$.* □

In a more general form, we have:

Theorem 2. $\mathcal{S} \models \mathcal{L} \Leftrightarrow (\mathcal{S}'||O'_{\mathcal{L}}) \models (l_{min} \rightsquigarrow l_{max})$. □

Theorem 2 indicates that the problem of checking whether a system model $\mathcal{S}$ satisfies an LSC requirement $\mathcal{L}$ can be equivalently transformed into a classical model checking problem ("ϕ leads-to φ") in Uppaal.

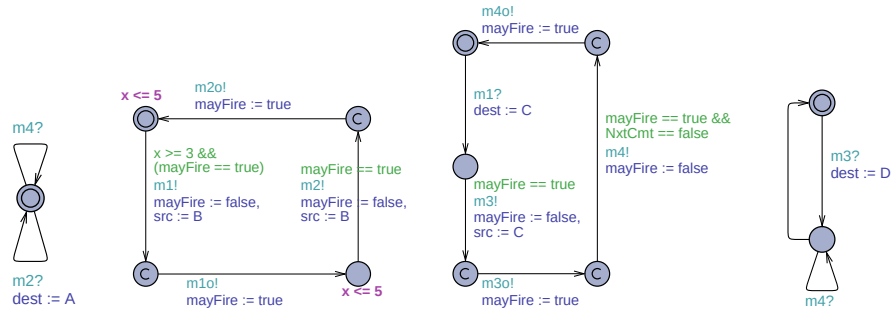
5 An example

We put things together by using the example in Fig. 1. The original system $\mathcal{S}$ consists of timed automata A, B, C, and D, having channels m_1, m_2, m_3, m_4, and clock variable x. The scenario-based requirement $\mathcal{L}$ is given in Fig. 1(e).

After modifying $\mathcal{S}$ and the translated observer TA $O_{\mathcal{L}}$, we get the newly composed network of TA ($\mathcal{S}'||O'_{\mathcal{L}}$), see Fig. 7 and Fig. 8.

For this newly composed model, we check in Uppaal the property ($l_{min} \rightsquigarrow l_{max}$), and it turns out to be satisfied. This indicates that $\mathcal{S}$ does satisfy the requirements that are specified in $\mathcal{L}$.

If in $\mathcal{L}$ the condition of m_2 is changed from $x \geq 2$ to e.g. $x \geq 4$, then the property will not be satisfied. There will be a counterexample, e.g., when $O'_{\mathcal{L}}$ has to synchronize on the channel m2o in location L2 of Fig. 8, but the value of clock x falls in $[3, 4)$, then it gets stuck there.



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

Fig. 7. The modified model $\mathcal{S}'$ of the original system in Fig. 1(a)-1(d).

6 Conclusions

This paper deals with the verification of real-time systems against scenario-based requirements by using model transformation and event spying techniques. Since both the LSC to TA translation and the non-intrusive composing method are automatic steps, our approach can be fully automated.

Based on previous work [17], the translation algorithms in this paper have been implemented as a prototype LSC-to-TA translator, which has been integrated into the Uppaal GUI and verification server. Experiments with some non-trivial examples showed the effectiveness of this method and tool.

Future work includes: (1) empirical evaluations for studying the applicability and scalability of this approach; (2) to support the translations of more chart elements such as subchart, if-then-else structure, loop, forbidden and ignored event, co-region, symbolic instances, and other chart modes; (3) to consider multiple charts for system modeling as well as for property specification; (4) to specify interaction scenarios for timed game solving and controller synthesis.

Acknowledgements. We thank Sandie Balaguer and Alexandre David for (re-)implementing the translation algorithms and tool, and integrating them into Uppaal. This research has received funding from the EuropeanCommunity's Seventh Framework Programme under grant agreement no. 214755.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. TCS, **126**: 183–235 (1994)
2. Alur, R., Holzmann, G.J., Peled, D.: An analyzer for message sequence charts. Software Concepts and Tools, **17**(2): 70–77 (1996)
3. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on Uppaal. In: SFM'04 (2004)
4. Bengtsson, J., Yi, W.: Timed Automata: Semantics, Algorithms and Tools. In: Lectures on Concurrency and Petri Nets 2003, Springer, 87-124 (2003)
5. Bontemps, Y.: Relating Inter-Agent and Intra-Agent Specifications: The Case of Live Sequence Charts. PhD thesis, University of Namur (2005)

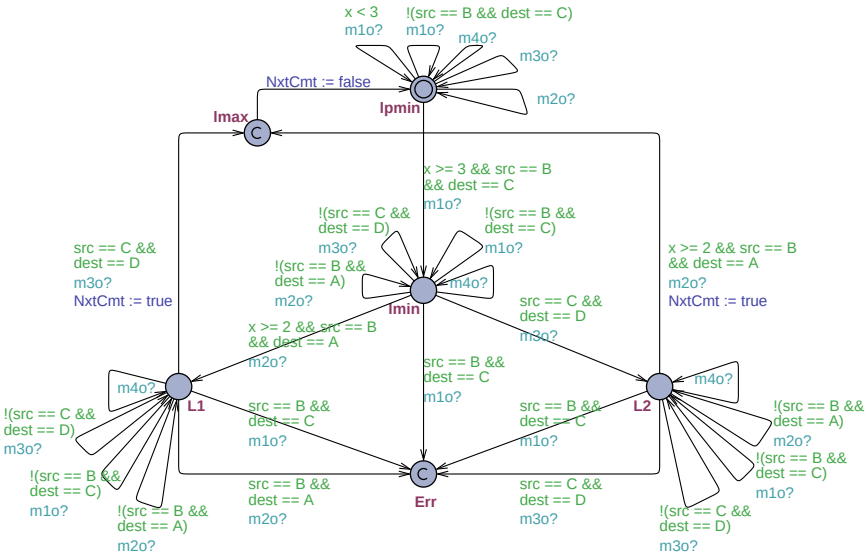


Fig. 8. The translated and modified observer TA $O'_{\mathcal{L}}$ of the chart in Fig. 1(e).

6. Bontemps, Y., Schobbens, P.-Y.: The computational complexity of scenario-based agent verification and design. J. Applied Logic **5**(2): 252–276 (2007).
7. Damm, W., Harel, D.: LSCs: Breathing Life into Message Sequence Charts. Formal Methods in System Design, **19**(1): 45–80 (2001)
8. Damm, W., Toben, T., Westphal, B.: On the Expressive Power of Live Sequence Charts. In: Program Analysis and Compilation 2006: 225–246 (2006)
9. Firley, T., Huhn, M., Diethers, K., Gehrke, T., Goltz, U.: Timed Sequence Diagrams and Tool-Based Analysis - A Case Study. In. Proc. UML'99: 645-660 (1999)
10. Genest, B., Minea, M., Muscholl, A., Peled, D.: Specifying and Verifying Partial Order Properties Using Template MSCs. In: Proc. FOSSACS'04: 195-210 (2004)
11. Harel, D., Marelly, R.: Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine. Spinger (2003)
12. Harel, D., Kugler, H.: Synthesizing State-Based Object Systems from LSC Specifications. Int. J. of Foundations of Computer Science, **13**(1), 5–51 (2002)
13. ITU: Z.120 ITU-TS Recommendation Z.120: Message Sequence Chart 2000. (1999)
14. Klose, J., Wittke, H.: An Automata Based Interpretation of Live Sequence Charts. In: TACAS'01, Springer, 512–527 (2001)
15. Kugler, H., Harel, D., Pnueli, A., Lu, Y., Bontemps, Y.: Temporal Logic for Scenario-Based Specifications. In: TACAS'05, Springer, 445–460 (2005)
16. Lahtinen, J.: Model checking timed safety instrumented systems. Research Report TKK-ICS-R3, Helsinki University of Technology, Espoo, Finland (2008)
17. Pusinskas S.: From Live Sequence Charts to Uppaal. PhD thesis (forthcoming)
18. Rye-Andersen J.G., Jensen M.W., Goettler R., Jakobsen M.: PEEL: Property Extraction Engine for LSCs. Master thesis, Aalborg University (2004)
19. Sengupta, B., Cleaveland, R.: Triggered Message Sequence Charts. In: FSE (2002)
20. Yovine S.: Kronos: A verification tool for real-time systems. STTT, **1**(1/2): 123-133, Springer-Verlag (1997)

Appendix: Proofs or lemmas and theorems

Lemma 1. If a configuration (c, v) of $\mathcal{L}$ corresponds to a semantic state (l, v) of $O_{\mathcal{L}}$, then: (1) each simregion s that follows (c, v) in $\mathcal{L}$ uniquely corresponds to an outgoing edge (l, l') in $O_{\mathcal{L}}$, and (2) the target configuration (c', v') of s in $\mathcal{L}$ uniquely corresponds to the target semantic state (l', v') in $O_{\mathcal{L}}$. □

Proof. For each simregion s in $\mathcal{L}$ that immediately follows (c, v) w.r.t. the partial order $\preccurlyeq'$ of $\mathcal{L}$, according to Section 3.3.3, s uniquely corresponds to an outgoing edge (l, l') from l in $O_{\mathcal{L}}$. Since the valuation function v is the same in (l, v) as in (c, v), and the condition in s is straightforwardly 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 straightforwardly 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 $\mathcal{L}$, then according to the ASAP semantics, s will be stepped over immediately from (c, v). Accordingly, the source location l is a committed location, 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). □

Theorem 1: For any trace tr in $O_{\mathcal{L}}$: $tr \models \mathcal{L} \Leftrightarrow (O_{\mathcal{L}}, tr) \models (l_{min} \rightsquigarrow l_{max})$. □

Proof. Let the initial cut of $\mathcal{L}$ be c_0. According to Section 3.3.2, c_0 corresponds to the initial location l_0 of $O_{\mathcal{L}}$. Since in the beginning all the clocks in $\mathcal{L}$ have the same initial values as in $O_{\mathcal{L}}$, the initial configuration (c_0, v_0) of $\mathcal{L}$ uniquely corresponds to the initial semantic state (l_0, v_0) of $O_{\mathcal{L}}$.

Only the legal behaviors (admissible traces) of $O_{\mathcal{L}}$ will be considered. We consider the following cases:

(1) $O_{\mathcal{L}}$ has only explicitly specified behaviors. By Lemma 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 $\mathcal{L}$ uniquely corresponds to the target semantic state (l', v') in $O_{\mathcal{L}}$. On the other hand, in (c_0, v_0) of $\mathcal{L}$, there could be a time delay d if and only if (l_0, v_0) of $O_{\mathcal{L}}$ can have the same time delay d.

By recursively applying Lemma 1 and the above result, we can conclude that any timed trace tr in $O_{\mathcal{L}}$ is also a timed trace in $\mathcal{L}$.

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

The reverse implication is proved similarly.

(2) $O_{\mathcal{L}}$ include behaviors of unconstrained events or cold violations. In this case, each unconstrained event m at a particular cut c in $\mathcal{L}$ uniquely corresponds

to an m?-labeled self-loop edge at a corresponding location l in $O_{\mathcal{L}}$, and each cold violation uniquely corresponds to an edge leading to l_{pmin} (if any) or l_{min} (otherwise). The two-way implications are proved similarly.

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

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

Lemma 2. If $O_{\mathcal{L}}$ has no committed location, and all $ch \in Chan$ are binary synchronization channels, then $\mathcal{S} \models \mathcal{L} \Leftrightarrow (\mathcal{S}'||O'_{\mathcal{L}}) \models (l_{min} \rightsquigarrow l_{max})$. □

Proof. Let $(\bar{l}, v)$ be a semantic state of the network of TA of $\mathcal{S}$, where $\bar{l}$ is a location vector, and v is the valuation of all clock variables. For each binary synchronization channel $ch \in Chan$, we have a transition $(\bar{l}, v) \xrightarrow{ch} (\bar{l}', v')$ if in two different processes of $\mathcal{S}$, there are two edges (l_i, l_{i+1}) and (l_j, l_{j+1}) labeled with ch! and 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 two 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 $\mathcal{S}$ and the observer TA $O_{\mathcal{L}}$ do not affect their legal (or admissible) behaviors (traces), i.e., the event notification mechanism and the locking mechanisms neither increase nor decrease the behaviors (traces) in $\mathcal{S}$ and $O_{\mathcal{L}}$. To this end, we prove that each synchronization in $\mathcal{S}$ uniquely corresponds to a pair of consecutive synchronizations in $(\mathcal{S}'||O'_{\mathcal{L}})$.

⇒):

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

If at a semantic state $(\bar{l}, v)$ of $\mathcal{S}$ there is a synchronization $(\bar{l}, v) \xrightarrow{ch} (\bar{l}', v')$, where $ch \in Chan$, we let the two coupling edges that carry ch! and ch? be (l_i, l_{i+1}) and (l_j, l_{j+1}), respectively. Clearly, they satisfy all the five requirements as listed above. According to the rules for modifying $\mathcal{S}$, edges (l_i, l_{i+1}) will correspond to two edges (l_i, l'_i) and (l'_i, l_{i+1}) in $\mathcal{S}'$, where l'_i is a newly added

committed location. Also according to the rules, the semaphore *mayFire* evaluates to false only when the current control is in a newly added committed location (see Fig. 5(d)). Now that the control is in l_i in $\mathcal{S}'$, the semaphore *mayFire* should evaluate to true. This together with the item $v \models g_i \wedge g_j$ in the above requirements indicate that the guards for (l_i, l_i') and (l_j, l_{j+1}) in $\mathcal{S}'$ to synchronize on ch are both satisfied. Besides, items 3-5 in the above 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 $\mathcal{S}'$ 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 $\mathcal{S}'$ will be immediately coupled with a corresponding edge in $O'_{\mathcal{L}}$. By the assumption $\mathcal{S} \models \mathcal{L}$, we know $O_{\mathcal{L}}$ does not restrict the behavior of $\mathcal{S}$ via its own conditions (e.g., via g_3 in Fig. 5(e)). This means that the cho-synchronization between $\mathcal{S}'$ and $O'_{\mathcal{L}}$ will not get stuck there due to the restrictions of $O'_{\mathcal{L}}$. Since after this synchronization, the clock variables in $\mathcal{S}'$ remain unchanged, we know that the location invariant Inv_{i+1} on l_{i+1} of $\mathcal{S}'$ will still be satisfied. After this synchronization, the two target locations in $\mathcal{S}'$ will be l_{i+1} and l_{j+1}, thus coinciding with the corresponding target locations l_{i+1} and l_{j+1} in $\mathcal{S}$. Therefore, we can conclude that given a trace tr in $\mathcal{S}$, there exists a unique trace tr' in $(\mathcal{S}'||O'_{\mathcal{L}})$ such that tr' and tr correspond.

By the definition of $\mathcal{S} \models \mathcal{L}$ (see Section 3.2), we know that if a timed trace μ in $\mathcal{S}$ arrives at the minimal cut of the main chart of $\mathcal{L}$, then μ must always be able to reach the maximal cut of that main chart. By Theorem 1 and Section 3.3, we know that if μ arrives at location l_{min} of $O'_{\mathcal{L}}$, then μ must always be able to reach location l_{max} of $O'_{\mathcal{L}}$.

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

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

$\Leftarrow$):

Similarly, we need to prove that each trace tr' in $(\mathcal{S}'||O'_{\mathcal{L}})$ uniquely corresponds to a trace tr in $\mathcal{S}$ such that tr' and tr are equivalent.

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

If $c \in Chan$, then after removing "*mayFire* == true" from the condition and removing "*mayFire* := false" from the assignment of the emitting edge (see Fig. 5(d)), the edge becomes exactly the corresponding edge in $\mathcal{S}$. 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 $\mathcal{S}$ can also fire.

If $c \in \{cho|ch \in Chan\}$, then the source location of the c!-emitting edge in $\mathcal{S}'$ must be a newly added committed location. This c! will be synchronized with a c?-receiving edge in $O'_{\mathcal{L}}$. And it will bring the control in $\mathcal{S}'$ from the committed location to the target location, which coincides with the corresponding target

location in $\mathcal{S}$. Due to the use of semaphore *mayFire*, no other synchronizations in $(\mathcal{S}'||O'_{\mathcal{L}})$ can preempt the execution of this c-synchronization.

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

Theorem 2. $\mathcal{S} \models \mathcal{L} \Leftrightarrow (\mathcal{S}'||O'_{\mathcal{L}}) \models (l_{min} \rightsquigarrow l_{max})$. □

Proof. This theorem is a generalization of Lemma 2 by canceling the restrictions.

If $ch \in Chan$ is a broadcast channel, the semantics of ch-synchronization is a little different. Since the modifications of the emitting edges in $\mathcal{S}$ do not affect the receiving edges in $\mathcal{S}$, we can still have a one-to-one mapping between the traces in $\mathcal{S}$ and in $(\mathcal{S}'||O'_{\mathcal{L}})$.

If there are committed locations in $O'_{\mathcal{L}}$, then we use the second semaphore *NxtCmt* to guarantee the non-interrupted execution at those committed locations in $O'_{\mathcal{L}}$. Since an edge (l, l') starting from a committed location l in $O'_{\mathcal{L}}$ represents an internal action (local) transition, it needs no synchronization with $\mathcal{S}'$. Thus the edge does not affect the behavior of $\mathcal{S}'$.

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