System Monitoring with Extended Message Sequence Chart (Extended Abstract)*

Ming Chai

Institut für Informatik
Humboldt Universität zu Berlin
ming.chai@informatik.hu-berlin.de

Abstract. Runtime verification is a lightweight formal verification technique that checks the correctness of the behaviour of a system. A problem with this technique is that most monitoring specification languages are not actually used in practice by system designers. To avoid this problem, we propose an monitoring approach on basis of an extension of live sequence charts (LSCs). We extend the standard LSCs as proposed by Damm and Harel by introducing the notation of "sufficient prechart", and by adding concatenation and product of charts. In this approach, a monitor solves the word problem that whether an observed behaviour of the underlying system is accepted by an extended LSC (eLSC) property. An on-line monitoring algorithm will be presented in the full version of the paper.

1 Intorduction

Runtime verification [3] is proposed for checking whether the behaviour of a system satisfies a correctness property. When compared to model checking and traditional testing, it is seen as a lightweight formal verification technique. Unlike model checking, runtime verification does not check all executions of the underlying system, but a finite trace. Therefore, it is able to avoid the so-called state explosion problem in model checking. When runtime verification is employed in the real system, it can be understand as ongoing testing. This makes the verification complete in a certain sense [4]. Such implementations are termed *online monitoring*, where a monitor checks the current execution of a system.
Runtime verification is performed by using a monitor. A monitor is device or a piece of software that consists of a correctness property and a checking algorithm. It reads an execution of the underlying system and reports whether the execution meets the property. An execution observed by a monitor is presented by a *trace*, which is a sequence of events. The correctness property of a monitor is typically specified by various temporal logic (e.g., linear temporal logic (LTL), metric temporal logic (MTL), time propositional temporal logic (TPTL) and first-order temporal logic (LTL^{FO})), regular expressions and context-free languages.

* This work was supported by the State Key Laboratory of Rail Traffic Control and Safety (Contract No.: RCS2012K001), Beijing Jiaotong University

Although these languages are expressiveness and technically sound for monitoring, they are not (yet) actually used in practice by system designers.
Graphical languages such as message sequence charts (MSCs) and UML sequence diagrams (UML-SDs) are widely used in industry for system specifications. Unfortunately, as semi-formal languages, the semantics of MSC and UML-SD is not defined formally. One of the central problems in these languages is that these languages cannot distinguish between necessary (i.e., enforced) and possible (i.e., allowed) behaviours. Since there does not seem to be an solution on this problem, these languages are not suitable for specifying monitoring correctness properties.
In this paper, we investigate the use of live sequence charts (LSCs) as proposed by Damm and Harel [2] for monitoring specifications. The LSC language is an extension of MSC. It specifies the exchange of messages among a set of instances. Using the notations of *universal* and *existential* chart, it can express that a behaviour of a system is necessary or possible. A universal chart specifies a necessary behaviour, whereas an existential chart specifies a possible behaviour.
For monitoring, we focus on universal charts. A universal chart typically consists of two components (*basic charts*): a *prechart* and a *main chart*. The intended meaning is that if the prechart is executed (i.e., the underlying system exhibits an execution which meets the prechart), then the main chart must be executed afterwards. The standard definition thus interprets the prechart as a necessary condition for the main chart.
For monitoring it is also important to express sufficient conditions of statements (e.g., absence of unsolicited responses). Unfortunately, sufficient conditions of statements cannot be expressed by a finite set of negation-free universal LSCs. Since the semantics of negative LSCs is hard to define, we extend LSCs to eLSCs by introducing the notion of a "sufficient" prechart for specifying this case. In contrast, we call the prechart of a standard universal chart a "*necessary*" prechart. With this extension, one can easily and intuitively express situations as above.
In our previous work, we made an assumption: every message appearing in an eLSC is unique. Although this assumption makes sense in practical work (e.g., every message is unique with a unique time stamp), it is considered to be very strong in theoretical work. In this extended abstract, we release this restriction in the eLSC language.
An eLSC based monitor essentially solves the well-known word problem: whether an observed trace is accepted by the language of an eLSC property. An on-line monitoring algorithm for this problem will be presented in the full version of the paper.

2 Definition of Basic Charts

We first define a basic chart of an eLSC.
A basic chart is visually similar to an MSC. It contains a set of messages and a set of lifelines. When a basic chart is executed, for each message two events occurs: the event of sending the message and the event of receiving it. The partial order of events induced by a basic chart is as follows.

– an event at a higher position in a lifeline precedes an event at a lower position in the same lifeline; and
– for each message m, the send-event of m precedes the receive-event of m.

Formally, a basic chart can be defined as follows.

Let Σ be a finite alphabet of messages m, i.e., $m \in \Sigma$. We use the labels $m.send$ and $m.recv$ to denote the event of sending a message m and the event of receiving m, respectively. We define $\widehat{\Sigma}^S \triangleq \{m.send | m \in \Sigma\}$ as the set of *send labels*, the set $\widehat{\Sigma}^R \triangleq \{m.recv | m \in \Sigma\}$ as the set of *receive lables*, and $\widehat{\Sigma} \triangleq \widehat{\Sigma}^S \cup \widehat{\Sigma}^R$ the set of *event labels*. A trace τ over $\widehat{\Sigma}$ is an element of $\widehat{\Sigma}^*$. The length of τ is $|\tau|$.

A lifeline l is a (possible empty) sequence of labels $l \triangleq (\hat{e}_1, ..., \hat{e}_n)$. A basic chart $\mathfrak{c}$ is a (possible empty) set of lifelines $\mathfrak{c} \triangleq \{l_1, ..., l_n\}$. Let $\mathbb{Z}_{\geq 0}$ be the set of positive integer. An event e in a basic chart is a tuple $e \triangleq (\hat{e}, x_1, x_2)$ with $x_1, x_2 \in \mathbb{Z}_{\geq 0}$, where x_1 is the index of the lifeline on which e occurs, and x_2 is the index of the position of the lifeline where e occurs. Each event of a basic chart is unique. We denote the set of events appearing in $\mathfrak{c}$ with $\mathscr{E}(\mathfrak{c})$. The set $\mathscr{E}(\mathfrak{c})$ can be partitioned into a set $\mathscr{S}$ of sending events and a set $\mathscr{R}$ of receiving events, i.e., $\mathscr{E}(\mathfrak{c}) = \mathscr{S} \cup \mathscr{R}$. Given a basic chart $\mathfrak{c}$ over an alphabet Σ, we define the following mappings.

1. a mapping $p : \mathscr{E}(\mathfrak{c}) \mapsto \mathbb{Z}_{\geq 0}$ that maps an event to the index of the lifeline on which it occurs;
2. a bijective mapping $f : \mathscr{S} \mapsto \mathscr{R}$ between sending and receiving events, matching a sending event to its corresponding receiving events.
3. a mapping $lab : \mathscr{E}(\mathfrak{c}) \mapsto \widehat{\Sigma}$ maps an event e to its label, i.e., $lab(e) \triangleq \hat{e}$.

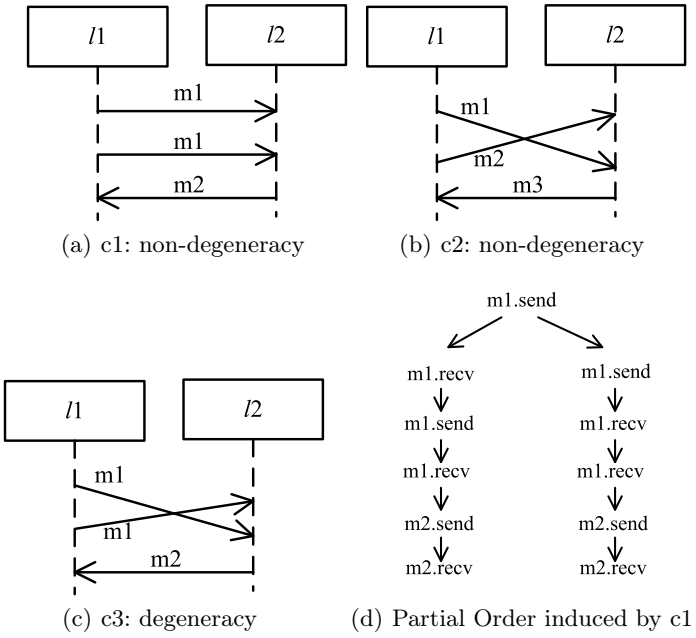
The chart $\mathfrak{c}$ induces a partial relation on $\mathscr{E}(\mathfrak{c})$ as follows.

1. for any $1 \leq xi \leq |\mathfrak{c}|$ and $1 \leq xj < |l_{xi}|$, it holds that $(\hat{e}, xi, xj) \prec (\hat{e}', xi, (xj + 1))$; and
2. for any $s \in \mathscr{S}$, it holds that $s \prec f(s)$.
3. $\prec$ is the smallest relation satisfying 1. and 2.

We admit the *non-degeneracy* assumption proposed by Alur et. al. [1]: a basic cannot reverse the receiving order of two identical messages sent by some lifeline. Formally, a basic chart is *degeneracy* if and only if there exist two sending events $e_1, e_2 \in \mathscr{S}$ such that $lab(e_1) = lab(e_2)$ and $e_1 \prec e_2$ and $f(e_1) \not\prec f(e_2)$. For instance, in fig. 1 (a) - (c), the basic charts c1 and c2 are non-degeneracy, whereas c3 is degeneracy. The basic chart c3 cannot distinguishes between the two receiving events of $m1$. The partial order of the labels induced by the basic chart c1 is shown as in fig 1(d). A set of traces over $\widehat{\Sigma}$ is defined by $\mathfrak{c}$ as follows:

$$\mathit{Traces}(\mathfrak{c}) \triangleq \{(lab(e_{x1}), lab(e_{x2}), ..., lab(e_{xn})) \mid \{e_{x1}, e_{x2}, ..., e_{xn}\} = \mathscr{E}(\mathfrak{c}); n = |\mathscr{E}(\mathfrak{c})|; \text{and for all } e_{xi}, e_{xj} \in \mathscr{E}(\mathfrak{c}), \text{if } e_{xi} \prec e_{xj}, \text{then } xi < xj\}.$$

Let $\widehat{\Sigma}_{\mathfrak{c}} \triangleq \bigcup_{e \in \mathscr{E}(\mathfrak{c})} \{lab(e)\}$ be the set of labels appearing in $\mathfrak{c}$. We call each $\sigma_{\mathfrak{c}} \in (\widehat{\Sigma} \backslash \widehat{\Sigma}_{\mathfrak{c}})$ a *stutter label*. For each basic chart $\mathfrak{c}$, the language $\mathcal{L}(\mathfrak{c})$ is defined by $\mathcal{L}(\mathfrak{c}) \triangleq \{(\sigma_{\mathfrak{c}}^*, e_1, \sigma_{\mathfrak{c}}^*, e_2, ..., \sigma_{\mathfrak{c}}^*, e_n, \sigma_{\mathfrak{c}}^*)\}$, where $(e_1\ e_2, ..., e_n) \in \mathit{Traces}(\mathfrak{c})$ and each $\sigma_{\mathfrak{c}}^*$ is a finite (or empty) sequence of stutter labels. A trace τ is *admitted* by a basic chart $\mathfrak{c}$ (denoted by $\tau \Vdash \mathfrak{c}$) if $\tau \in \mathcal{L}(\mathfrak{c})$.



(a) c1: non-degeneracy (b) c2: non-degeneracy

(c) c3: degeneracy (d) Partial Order induced by c1

Fig. 1. Example:Degeneracy in Basic Charts

3 Extended LSCs

A universal chart consists of two basic charts: a *prechart* and a *main chart* (drawn within a solid rectangle). For an eLSCs, there are two possibilities of the prechart: a necessary prechart (drawn within surrounding hashed hexagons) or a sufficient prechart (dran within surrounding solid hexagons). Formally, the syntax of eLSCs is as follows.

Definition 1. *An eLSC is a tuple* $\mathfrak{u} \triangleq (\mathfrak{p}, \mathfrak{m}, \text{Cond})$*, where* $\mathfrak{p}$ *and* $\mathfrak{m}$ *are a prechart and a main chart, and* $\text{Cond} \in \{Nec, Suff\}$ *denotes if* $\mathfrak{p}$ *is a necessary or sufficient prechart.*

We define an althernative semantics of eLSCs. On one hand, an eLSC with a necessary prechart intuitively specifies all traces composed of two segments such that, if the first segment is admitted by the prechart, then the second must be admitted by the main chart. On the other hand, an eLSC with a sufficient prechart specifies all traces composed of two segments such that, the first segment cannot be admitted by the prechart, unless the second is admitted by the main chart.
Given an eLSC $\mathfrak{u} = (\mathfrak{p}, \mathfrak{m}, Cond)$, the stutter labels of $\mathfrak{u}$ are $\sigma_{\mathfrak{u}} \in \left(\widehat{\Sigma} \backslash (\widehat{\Sigma}_{\mathfrak{p}} \cup \widehat{\Sigma}_{\mathfrak{m}})\right)$. The language $\mathcal{L}(\mathfrak{p})$ of the prechart (resp. the language $\mathcal{L}(\mathfrak{m})$ of the main chart) is defiend with $Traces(\mathfrak{p})$ (resp. $Traces(\mathfrak{m})$) and these stutter labels as above.
For languages $\mathcal{L}$ and $\mathcal{L}'$, let $(\mathcal{L} \circ \mathcal{L}')$ be the concatenation of $\mathcal{L}$ and $\mathcal{L}'$ (i.e., $(\mathcal{L} \circ \mathcal{L}' \triangleq \{(\tau\tau') \mid \tau \in \mathcal{L} \ \ and \ \ \tau' \in \mathcal{L}')$); and $\overline{\mathcal{L}}$ be the complement

of $\mathcal{L}$ (i.e., for any $\tau \in \widehat{\Sigma}^*$, it holds that $\tau \in \overline{\mathcal{L}}$ iff $\tau \notin \mathcal{L}$). The alternative semantics of eLSCs is defined as follows.

Definition 2. *Given a finite alphabet Σ, the language of an eLSC $\mathfrak{u} \triangleq (\mathfrak{p}, \mathfrak{m}, Cond)$ is*
$\mathcal{L}(\mathfrak{u}) \triangleq \overline{\mathcal{L}(\mathfrak{p}) \circ \overline{\mathcal{L}(\mathfrak{m})}}$*, if $Cond = Nec$; and*
$\mathcal{L}(\mathfrak{u}) \triangleq \overline{\overline{\mathcal{L}(\mathfrak{p})} \circ \mathcal{L}(\mathfrak{m})}$*, if $Cond = Suff$.*

This formalizes the intuitive interpretation given above. An eLSC specification $\mathfrak{U}$ is a finite set of eLSCs. The language of $\mathfrak{U}$ is $\mathcal{L}(\mathfrak{U}) \triangleq \bigcap_{\mathfrak{u} \in \mathfrak{U}} \mathcal{L}(\mathfrak{u})$.

4 Concatenations of eLSCs

Concatenation of two eLSCs essentially introduces partial orders of executions of the charts. This feature can be inherited by eLSC specifications.
We first define the concatenation of basic charts $\mathfrak{c}$ and $\mathfrak{c}'$, denoted with $(\mathfrak{c} \to \mathfrak{c}')$. Intuitively, a trace composed of two segments υ and υ' is in the language of $(\mathfrak{c} \to \mathfrak{c}')$ if and only if υ and υ' are admitted by $\mathfrak{c}$ and $\mathfrak{c}'$, respectively. Formally, the language of $(\mathfrak{c} \to \mathfrak{c}')$ is

$$\mathcal{L}(\mathfrak{c} \to \mathfrak{c}') \triangleq \left(\mathcal{L}(\mathfrak{c}) \cap \mathcal{L}(\mathfrak{c}') \cap \overline{\overline{\mathcal{L}(\mathfrak{c})} \circ \mathcal{L}(\mathfrak{c}')}\right).$$

Since an eLSC $\mathfrak{u}$ consists of two basic charts $\mathfrak{p}$ and $\mathfrak{m}$, there are four possibilitites to define the ocncatenation of eLSCs $\mathfrak{u}$ and $\mathfrak{u}'$: $(\mathfrak{p} \to \mathfrak{p}')$, $(\mathfrak{p} \to \mathfrak{m}')$, $(\mathfrak{m} \to \mathfrak{p}')$ and $(\mathfrak{m} \to \mathfrak{m}')$.
The concatenation of $\mathfrak{u}$ and $\mathfrak{u}'$ is defined to be a tuple $\delta \triangleq (\mathfrak{u}, \mathfrak{u}', Mode)$, where $Mode \in \{pp, pm, mp, mm\}$. Formally, the semantics of δ is given as follows.

Definition 3. *Given two eLSCs $\mathfrak{u}$ and $\mathfrak{u}'$, the language of $\delta \triangleq (\mathfrak{u}, \mathfrak{u}', Mode)$ is*

$$\mathcal{L}(\delta) \triangleq \left(\mathcal{L}(\mathfrak{u}) \cap \mathcal{L}(\mathfrak{u}') \cap \overline{\overline{\mathcal{L}(\mathfrak{c})} \circ \mathcal{L}(\mathfrak{c}')}\right), \text{ where}$$

$\mathfrak{c} = \mathfrak{p}$ *and* $\mathfrak{c}' = \mathfrak{p}'$*, if $Mode = pp$;*
$\mathfrak{c} = \mathfrak{p}$ *and* $\mathfrak{c}' = \mathfrak{m}'$*, if $Mode = pm$;*
$\mathfrak{c} = \mathfrak{m}$ *and* $\mathfrak{c}' = \mathfrak{p}'$*, if $Mode = mp$; and*
$\mathfrak{c} = \mathfrak{m}$ *and* $\mathfrak{c}' = \mathfrak{m}'$*, if $Mode = mm$.*

It can be shown that the language of δ is the same as the language of the eLSC specification $\{\mathfrak{u}, \mathfrak{u}', (\mathfrak{c}, \mathfrak{c}', \mathit{suff})\}$, where the values of $\mathfrak{c}$ and $\mathfrak{c}'$ are assigned according to $Mode$.

5 Product of eLSCs

We recall the definition of the *product* $\mathcal{L}||\mathcal{L}'$ of two languages $\mathcal{L}$ and $\mathcal{L}'$ of traces.
A label transition system is a tuple $S \triangleq (Q, q^{ini}, q^{fin}, \widehat{\Sigma}, \mathfrak{R})$, where

- Q is a (non-empty) finite set of *states*;
- $q^{ini} \in S$ is the *initial state*;
- $q^{fin} \in S$ is the *final state*;
- $\widehat{\Sigma}$ is a finite set of labels; and
- $\mathfrak{R} \subseteq Q \times \widehat{\Sigma} \times Q$ is a finite set of binary relations. Each triple $(q, e, q') \in \mathfrak{R}$ is called a *transition*.

Given a transition $\mathfrak{r} = (q, e, q') \in \mathfrak{R}$, we write $\mathsf{A}(\mathfrak{r}) \triangleq e$ for the label of $\mathfrak{r}$, and we write $\mathsf{Pre}(\mathfrak{r}) \triangleq q$ and $\mathsf{Post}(\mathfrak{r}) \triangleq q'$ for the predecessor state and successor state of $\mathfrak{r}$, respectively. A finite trace $\tau \triangleq (e_1, ...e_n)$ over $\widehat{\Sigma}$ is accepted by S (denoted with $\tau \models S$) if and only if there exists a sequence of transitions $(\mathfrak{r}_1, ..., \mathfrak{r}_n)$ such that

- $\mathsf{Pre}(\mathfrak{r}_1) = q^{ini}$, $\mathsf{Post}(\mathfrak{r}_m) = q^{fin}$ and for all $0 \leq i < m$ it hold that $\mathsf{Post}(\mathfrak{r}_i) = \mathsf{Pre}(\mathfrak{r}_{i+1})$; and
- $(\mathsf{A}(\mathfrak{r}_1), ..., \mathsf{A}(\mathfrak{r}_m)) = \tau$.

We say a label transition system is *linear* if and only if for all $\mathfrak{r}, \mathfrak{r}' \in \mathfrak{R}$ it holds that $\mathsf{Post}(\mathfrak{r}) \neq \mathsf{Pre}(\mathfrak{r}')$, $\mathsf{Post}(\mathfrak{r}) \neq \mathsf{Post}(\mathfrak{r}')$ and $\mathsf{Pre}(\mathfrak{r}) \neq \mathsf{Pre}(\mathfrak{r}')$. It can be shown that there exists only one trace accepted by a linear label transition system. For a linear label transition system S and the trace τ accepted by S, we define $\mathsf{LLTS}(\tau) \triangleq S$. The product of two traces τ_1 and τ_2 can then be defined with the product of $\mathsf{LLTS}(\tau_1)$ and $\mathsf{LLTS}(\tau_2)$.
Now we define the product $S_1||S_2$ of two label transition systems S and S'.

Definition 4. *Given two label transition systems* $S_1 \triangleq (Q_1, q_1^{ini}, q_1^{fin}, \widehat{\Sigma}_1, \mathfrak{R}_1)$ *and* $S_2 \triangleq (Q_2, q_2^{ini}, q_2^{fin}, \widehat{\Sigma}_2, \mathfrak{R}_2)$*, the product* $S = (S_1||S_2)$ *is a tuple* $(Q, q^{ini}, q^{fin}, \widehat{\Sigma}, \mathfrak{R})$*, where*

- $Q \triangleq Q_1 \times Q_2$*;*
- $q^{ini} \triangleq (q_1^{ini}, q_2^{ini})$*;*
- $q^{fin} \triangleq (q_1^{fin}, q_2^{fin})$*;*
- $\widehat{\Sigma} \triangleq \widehat{\Sigma}_1 \cup \widehat{\Sigma}_2$*;*
- $((q_1, q_2), e, (q_1', q_2')) \in \mathfrak{R}$ *iff*
  - $(q_1, e, q_1') \in \mathfrak{R}_1$ *and* $(q_2, e, q_2') \in \mathfrak{R}_2$*, or*
  - $(q_1, e, q_1') \in \mathfrak{R}_1$ *and* $\forall q_2' \in Q_2 : (q_2, e, q_2') \notin \mathfrak{R}_2$ *and* $q_2 = q_2'$*, or*
  - $(q_2, e, q_2') \in \mathfrak{R}_2$ *and* $\forall q_1' \in Q_1 : (q_1, e, q_1') \notin \mathfrak{R}_1$ *and* $q_1 = q_1'$*.*

We write $\mathcal{L}(S_1||S_2) \triangleq \{\tau \mid \tau \models (S_1||S_2)\}$ for the set of trace accepted by $(S_1||S_2)$. Given $S_1 \triangleq \mathsf{LLTS}(\tau_1)$ and $S_2 \triangleq \mathsf{LLTS}(\tau_2)$, the product of τ_1 and τ_2 is $\tau_1||\tau_2 \triangleq \mathcal{L}(S_1 \mid\mid S_2)$.
The product of two languages $\mathcal{L}_1$ and $\mathcal{L}_2$ is $\mathcal{L}_1||\mathcal{L}_2 \triangleq \bigcup_{\tau_1 \in \mathcal{L}_1, \tau_2 \in \mathcal{L}_2} (\tau_1||\tau_2)$.
The language of the product $(\mathfrak{c}_1||\mathfrak{c}_2)$ of basic charts $\mathfrak{c}_1$ and $\mathfrak{c}_2$ is defined as follows.

Definition 5. *Given two basic charts* $\mathfrak{c}$ *and* $\mathfrak{c}_2$ *over* $\widehat{\Sigma}$ *and the stutter label* $\sigma \in \left(\widehat{\Sigma} \backslash (\widehat{\Sigma}_{\mathfrak{c}_1} \cup \widehat{\Sigma}_{\mathfrak{c}_2})\right)$*, the language of* $\mathfrak{c}_1||\mathfrak{c}_2$ *is*

$$\mathcal{L}(\mathfrak{c}_1||\mathfrak{c}_2) \triangleq \{(\sigma^*, e_1, \sigma^*, e_2, ..., \sigma^*, e_n, \sigma^*) \mid (e_1, e_2, ..., e_n) \in (\mathit{Traces}(\mathfrak{c}_1) \mid\mid \mathit{Traces}(\mathfrak{c}_2))\}$$

Similar as concatenations, there are four possibilities to define the product of eLSCs $\mathfrak{u}_1$ and $\mathfrak{u}_2$: $\mathfrak{p}_1||\mathfrak{p}_2$, $\mathfrak{p}_1||\mathfrak{m}_2$, $\mathfrak{m}_1||\mathfrak{p}_2$ and $\mathfrak{m}_1||\mathfrak{m}_2$.

Definition 6. *The product of two eLSCs* $\mathfrak{u}_1$ *and* $\mathfrak{u}_2$ *is defined to be a tuple* $\mathfrak{u}_1||\mathfrak{u}_2 \triangleq (\mathfrak{u}_1, \mathfrak{u}_2, \mathit{Prod})$*, where* $\mathit{Prod} \in \{p||p, p||m, m||p, m||m\}$.

Formally, the semantics of the four product possibilities is given as follows.

Definition 7. *Given two eLSCs* $\mathfrak{u}_1$ *and* $\mathfrak{u}_2$*, the language of the product of* $\mathfrak{u}_1$ *and* $\mathfrak{u}_2$ *is* $\mathcal{L}(\mathfrak{u}_1||\mathfrak{u}_2) \triangleq (\mathcal{L}(\mathfrak{u}_1) \cap \mathcal{L}(\mathfrak{u}_2) \cap \mathcal{L}(\mathfrak{c}_1||\mathfrak{c}_2))$*, where*
$\mathfrak{c} = \mathfrak{p}$ *and* $\mathfrak{c}' = \mathfrak{p}'$*, if* $\mathit{Prod} = \mathrm{p||p}$*;*
$\mathfrak{c} = \mathfrak{p}$ *and* $\mathfrak{c}' = \mathfrak{m}'$*, if* $\mathit{Prod} = \mathrm{p||m}$*;*
$\mathfrak{c} = \mathfrak{m}$ *and* $\mathfrak{c}' = \mathfrak{p}'$*, if* $\mathit{Prod} = \mathrm{m||p}$*; and*
$\mathfrak{c} = \mathfrak{m}$ *and* $\mathfrak{c}' = \mathfrak{m}'$*, if* $\mathit{Prod} = \mathrm{m||m}$.

References

1. Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of Message Sequence Charts. *Software Engineering, IEEE Transactions on*, 29(7):623–633, 2003.
2. Werner Damm and David Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001.
3. Klaus Havelund and Grigore Roşu. Monitoring Java Programs with Java PathExplorer. *Electronic Notes in Theoretical Computer Science*, 55(2):200–217, 2001.
4. Martin Leucker and Christian Schallhart. A Brief Account of Runtime Verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.