Monitoring with Parametrized Extended Life Sequence Charts*

Ming Chai[1] and Bernd-Holger Schlingloff[2]

[1] [2] Humboldt Universität zu Berlin
[2] Fraunhofer FOKUS
{ming.chai, hs}@informatik.hu-berlin.de

Abstract. Runtime verification is a lightweight verification technique that checks whether an execution of a system satisfies a given property. A problem in monitoring specification languages is to express parametric properties, where the correctness of a property depends on both the temporal relations of events, and the data carried by events. In this paper, we introduce parametrized extended live sequence charts (PeLSCs) for monitoring sequences of data-carrying events. The language of PeLSCs is extended from life sequence charts by introducing condition and assignment structures. We develop a translation from PeLSCs into the hybrid logic **HL**, and prove that the word problem of the PeLSCs is linear with respect to the size of a parametrized event trace. Therefore, the formalism is feasible for on-line monitoring.

1 Introduction

Even with most advanced quality assurance techniques, correctness of complex software can never be guaranteed. To solve this problem, *runtime verification* has been proposed to provide on-going protection during the operational phase. Runtime Verification checks whether an execution of a computational system satisfies or violates a given correctness property. It is performed by using a *monitor*. This is a device or a piece of software that observes the system under monitoring (SuM) and generates a certain verdict (*true* or *false*) as the result. Compared to model checking and testing, this technique is considered to be a lightweight validation technique, since it does not try to cover all possible executions of the SuM. It detects failures of an SuM directly in its actual running environment. This avoids some problems of other techniques, such as imprecision of the model in model checking, and inadequateness of the artificial environment in testing.

An execution of a computational system checked by a monitor can be formalized by a sequence of events. One of the challenges in building a runtime verification system is to define a suitable specification language for monitoring

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

properties. A monitoring specification language should be *expressive* and *attractive* [22]: The language should be able to express all expected monitoring properties, and the language should keep the formulations simple for simple properties. A simple formulation means that the size of the formulation is small, and the notations of the formulation is understood by users (e.g., system designers).

Over the last years, various runtime verification systems have been developed which use some form of temporal logic, including linear temporal logic (LTL), metric temporal logic (MTL), time propositional temporal logic (TPTL) and first-order temporal logic (LTL^{FO}). Although these specification languages are expressive and technically sound for monitoring, software engineers are not familiar with them and need extensive training to use them efficiently. Therefore, many runtime verification systems support also other specification languages, such as regular expressions and context-free grammars. Unfortunately, it is difficult to specify properties for parallel systems in these languages, and they are not (yet) used in practice by system designers.

In previous work [14], we proposed an extension of live sequence chart (LSC) [18] for expressing monitoring properties. LSC is a visual formalism that specifies the temporal relations of the exchange of messages among instances. It extends the classical message sequence chart formalism (MSC) by introducing possible and mandatory elements, including universal and existential charts, and hot and cold messages and conditions. With these extensions, LSCs are able to distinguish between required and allowed behaviours of an SuM. Our language of the proposed extended LSCs (eLSCs) introduces modal pre-charts. That is, we distinguish between pre-charts that are necessary conditions of main-charts and those that are sufficient conditions of main-charts.

The eLSC-based monitoring approach so far can not handle parametric properties, where the correctness of a property depends on both the temporal relations of events and data carried by the events. One possible workaround for this shortage is to formalize each assignment of data with a unique atomic proposition. However, since the domain of data can be infinite or unknown, this approach is not sufficient in general. We extend eLSC to *parametrized eLSC* (PeLSC) by introducing *assignment structures* and *condition structures*.

In this paper, we model data-carrying events with *parametrized events*, where the data is represented by parameters. Consider a client/server system that allows clients to access a server, and consider the following properties.

(**P1**)*: If there is a log in to the server, it must be followed by a log out.*
(**P2**)*: A log out event can not occur, unless it is preceded by a log in.*
(**P3**)*: If a client logs in to the server, it must log out within 200 sec.*

For the first two properties, the monitor can observe a propositional events *login* and *logout*. The expected behaviours can be formalized by the following regular expressions L1 and L2, respectively.

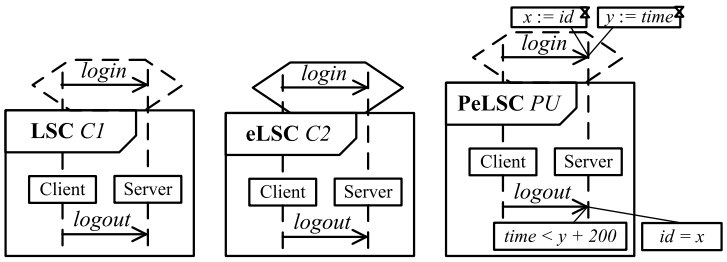
$$\mathbf{L1} \triangleq \overline{\Sigma^* \circ \{login\} \circ \overline{\{logout\} \circ \Sigma^*}}$$
$$\mathbf{L2} \triangleq \overline{\overline{\Sigma^* \circ \{login\}} \circ \{logout\} \circ \Sigma^*}$$

For the property (**P3**), each of the *login* and *logout* events carries a client name and a time stamp. An execution of this system can be formalized by a sequence of parametrized events. Each of the propositional events carries two parameters *client_id* (`id`) and *time_stamp* (`time`). With these definitions, property (P3) can be written more formally as follows:

> When the system emits a *login* event with ($\texttt{id} = x$) and ($\texttt{time} = y$), a *logout* event with ($\texttt{id} = x'$) and ($\texttt{time} = y'$) should occur afterwards, where ($x' = x$) and ($y' \leq (y + 200)$).

The PeLSCs of Fig. 1 specify the three properties above. The chart ($C1$) is a standard LSC formalizing (P1). Property (P2) cannot be formalized with LSCs; an eLSC for it is ($C2$). Property (P3) involves parameters on infinite or unknown domains and thus cannot be expressed by eLSCs. A PeLSC for it is (PU). Formal definitions being given below, we note that the language PeLSC contains variables, assignment and conditions for dealing with data-parametrized events. An assignment structure is used to store an arbitrary parameter value, and a condition structure is used to express constraints on such values. With these extensions, PeLSCs can be used for monitoring systems where events carry data.

To generate monitors, we translate PeLSCs into (a subclass of) the hybrid logic (**HL**) [13]. **HL** has a type of symbols called *nominals* that represent *names* of parameters. Let s be a symbol, an **HL** formula may contain the *downarrow* binder "$x \downarrow s$.". When evaluating an **HL** formula over a parametrized event trace, the downarrow binder assigns all variables x in the formula to the value of the parameter s of the "current" parametrized event.



(a) PeLSC for **P1** (b) PeLSC for **P2** (c) PeLSC for **P3**

Fig. 1. Examples: PeLSCs for properties of a client/server system

A monitor essentially solves the word-problem: given a trace, decide whether the trace is in the language defined by a monitoring property. As a main result of this paper, we prove that the complexity of the word-problem of PeLSCs is linear if the propositions in the condition structures express only comparisons of parameter values. Thus, monitoring can be done on-line, while the SuM is running.

The rest of the paper is organized as follows. Section 2 outlines related work. Section 3 introduces parametrized eLSCs (PeLSCs), including their syntax and trace-based semantics. Section 4 presents a translation from PeLSCs into a subclasss of **HL**, and proves the complexity of the word problem of PeLSCs. Section 5 contains some conclusions and hints for future work.

2 Related Work

Our extension of LSCs is inspired by the treatment of time in live sequence charts proposed by Harel et. al. [21]. There, a time constraint in LSCs is defined by a combination of assignment structures and condition structures. In contrast, we provide a more general notation for arbitrary data parameters.

There are several other runtime verification approaches for handling parametrized events. The EAGLE logic [5], which is a linear μ-calculus, is one of the first logics in runtime verification for specifying and monitoring data-relevant properties. Although EAGLE has rich expressiveness, it has high computational costs [7]. To avoid this problem, other rule-based methods have been introduced. They are based on MetateM [4] and the Rete algorithm [19]. MetateM provides a framework of executing temporal formulae and Rete is an efficient algorithm for matching patterns with objects. Inspired by MetateM, RuleR is an efficient rule-based monitoring system that can compile various temporal logics [7]. LogicFire is an internal domain specification language for artificial intelligence on basis of Rete [23]. The rule-based runtime verification systems have high performance. However, their implementations are still complex. The language of PeLSCs has a comparable expressiveness. However, the implementation of PeLSC based runtime verification system is easier because monitors are generated automatically with the translation algorithm.

TraceMatches [1] is essentially a regular expression language. It extends the language of AspectJ [24] by introducing free variables in the matching patterns. TraceContract is an API for trace analysis, implemented in Scala, which is able to express parametric properties with temporal logic [6]. Monitoring oriented programming (MOP) is an efficient and generic monitoring framework that integrates various specification languages [16]. In particular, JavaMOP deals with parametric specification and monitoring using TraceMatches [25]. TraceMatches and JavaMOP are defined on the basis of *trace slicing*, which translates parametrized events into propositional events. With trace slicing, the problem of checking parametrized event traces is translated into a (standard) propositional word problem. Although JavaMOP has high performance, to our opinion its expressiveness is insufficient. As pointed out in [3], trace slicing can only

handle traces where all events with the same name carry the same parameters. Our PeLSCs based approach overcomes this shortage by using formula rewriting algorithms.

Another important direction of parametric monitoring is based on automata theory. Quantified event automata [3] are an extension of the trace slicing methods mentioned above. They are strictly more expressive than TraceMatches. In that context, data automata have been proposed as a down-scaled version of Rete to an automaton-based formalism [22]. Unfortunately, properties of parallel systems have complex formulations when expressed by automata. PeLSCs can keep the monitoring specification attractive when dealing with such properties.

Various extensions of LTL have been proposed for parametric monitoring. If time is the only parameter, properties can be formalized with real-time logics such as TLTL [12], MTL [10] and TPTL [15]. For other parameters, first order extensions of LTL have been introduced. Parametrized LTL [28] contains a binary binding operator, and is further translated into parametrized automata for monitoring. First-order temporal logic LTL^{FO} includes both first-order and temporal connectives [26]. For monitoring LTL^{FO} an algorithm using a spawning automaton has been developed [11]. However, the word problem of LTL^{FO} is PSPACE-complete [11], and the translation has a potential of suffering from the state explosion problem. A domain-specific language for monitoring the exchange of XML messages of web service is LTL^{FO+} [20]. This language has a lower complexity than full first order temporal logic. However, its expressiveness is limited by only allowing to express equivalence of variables. Metric Temporal First-order Logic (MFOTL) adds quantifiers to MTL [9], and has been used for monitoring data applications[8]. An MFOTL monitoring system has been built based on a trace decomposing technique, which may introduce additional errors/mistakes. Similiar to the languages of automata, all these temporal logics have difficulties in specifying concurrency properties. The language of PeLSCs can avoid these shortcomings. The word problem for PeLSCs is linear with respect to the size of traces. Meanwhile, PeLSCs have richer expressiveness than LTL^{FO+} by allowing to express general comparisons of terms. Our rewriting based algorithms avoid the problems introduced by the LTL to automata translations, and the trace decomposing techniques.

3 Definitions of Parametrized eLSCs

This section presents the syntax and semantics of *parametrized extended LSCs* (PeLSCs). The PeLSCs are interpreted over *parametrized event traces*, which are defined as follows.

Let $\Sigma \triangleq \{e_1, e_2, ..., e_n\}$ be a finite alphabet of *events*, $\mathcal{N} \triangleq \{s_1, s_2, ...\}$ be a countable set of *nominals* and $\mathcal{D} \triangleq \{d_1, d_2, ...\}$ any *domain* (e.g., integers, strings, or reals). A *parameter* is a pair $p \triangleq \langle s, d \rangle$ from $\mathcal{N} \times \mathcal{D}$, where s is the *name* of p and d is the *value* of p.

Definition 1 (Parametrized event). *Given an alphabet Σ of events, a set $\mathcal{N}$ of nominals and a domain $\mathcal{D}$, a* parametrized event *is a pair $\mathfrak{w} \triangleq \langle e, \mathcal{P} \rangle$, where $e \in \Sigma$ is an event and $\mathcal{P} \in 2^{\mathcal{N} \times \mathcal{D}}$ is a set of parameters.*

Given a parametrized event $\mathfrak{w}$ with $\mathcal{P} \triangleq \{\langle s_1, d_1 \rangle, ..., \langle s_m, d_m \rangle\}$, we define $\mathsf{Evet}(\mathfrak{w}) \triangleq e$, $\mathsf{Para}(\mathfrak{w}) \triangleq \mathcal{P}$ and $\mathsf{Nam}(\mathfrak{w}) \triangleq \{s_1, ..., s_m\}$. A parametrized event $\langle e, \mathcal{P} \rangle$ is *deterministic* if each parameter name in $\mathcal{P}$ is unique, i.e., for all $p, p' \in \mathcal{P}$ it holds that $s \neq s'$. In this paper, we assume that all parametrized events are deterministic.

Parametrized event traces basically are finite sequences of parameterized events.

Definition 2 (Parametrized event trace). *Given $\mathcal{N}$ and $\mathcal{D}$, a parameter trace $\rho \triangleq (\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_n)$ over $\mathcal{N} \times \mathcal{D}$ is a finite sequence of sets of parameters, i.e., an element of $(2^{\mathcal{N} \times \mathcal{D}})^*$. Given Σ, $\mathcal{N}$ and $\mathcal{D}$, a parametrized event trace $\tau \triangleq \langle \sigma, \rho \rangle$ is a pair of a finite event trace σ and a parameter trace ρ with the same length, i.e., $\sigma \in \Sigma^*$ and $\rho \in (2^{\mathcal{N} \times \mathcal{D}})^*$ and $|\sigma| = |\rho|$.*

By $\tau[i] \triangleq \langle \sigma[i], \rho[i] \rangle$ we denote the i^{th} parametrized event of τ, where $\sigma[i]$ and $\rho[i]$ are the i^{th} element of σ and ρ, respectively.

3.1 Syntax of PeLSCs

A universal PeLSC consists of two basic charts: a *pre-chart* and a *main-chart*. A basic chart is visually similar to an MSC. It specifies the exchange of messages among a set of instances. Each instance is represented by a lifeline. Lifelines in a basic chart are usually drawn as vertical dashed lines, and messages are solid arrows between lifelines. For each message, there are two actions: the action of sending the message and the action of receiving it. Each action occurs at a unique position in a lifeline. The partial order of actions induced by a basic chart is as follows.

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

Formally, we define basic charts as follows.

Let M be a set of *messages*, and let the set of events be given as $\Sigma \triangleq (M \times \{!, ?\})$. That is, an event e is either $m!$ (indicating that message m is sent, or $m?$ (indicating that m is received).

A *lifeline* l is a finite (possibly empty) sequence of events $l \triangleq (e_1, e_2, ..., e_n)$. A *basic chart* C is an n-tuple of lifelines $\langle l_1, ..., l_n \rangle$ with $l_i = (e_{i1}, ..., e_{im})$. We say that an event e *occurs* at the location (i, j) in chart C if $e = e_{ij}$. An *event occurrence* $\mathfrak{o} \triangleq (e, i, j)$ is a tuple consisting of an event e and the location of e. We define $loc(\mathfrak{o}) \triangleq (i, j)$ as the location of an event occurrence o, and $lab(\mathfrak{o}) \triangleq e$ be the event of o. We denote the set of event occurrences appearing in C with $\mathsf{EO}(C)$ A *communication* $\langle (m!, i, j), (m?, i', j') \rangle$ in C is a pair of two

event occurrences in C representing sending and receiving of the same message m. We define $mat(m!, i, j) \triangleq (m?, i, j)$ to match a sending event occurrence to a receiving event occurrence of the same communication. A communication does not have to be completely specified by a basic chart. That is, it is possible that only the sending event or the receiving event of a message appears in a basic chart. In addition, an event is allowed to occur multiple times in a basic chart, i.e., a basic chart can express that a message is repeatedly exchanged. However, each event occurrence is unique in a basic chart.

The partial relation induced by a chart C on $\mathsf{EO}(C)$ is formalized as follows.

1. for any $1 \leq xj < |l_{xi}|$ with l_{xi} being a lifeline in C, it holds that $(e, xi, xj) \prec (e', xi, (xj + 1))$;
2. for any $\mathfrak{o} \in S$, it holds that $\mathfrak{o} \prec mat(\mathfrak{o})$; and
3. $\prec$ is the smallest relation satisfying 1 and 2.

We admit the *non-degeneracy* assumption proposed by Alur et. al. [2]: a basic chart 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 event occurrences $\mathfrak{o}_1, \mathfrak{o}_2 \in S$ with $\mathfrak{o}_1 \prec \mathfrak{o}_2$ such that $lab(\mathfrak{o}_1) = lab(\mathfrak{o}_2)$ and $mat(\mathfrak{o}_1) \not\prec mat(\mathfrak{o}_2)$.

For a basic chart, event occurrences are allowed to be absent, i.e., it is possible that only a sending event or a receiving event of a message appears in a basic chart. Each event occurrence is unique in a basic chart.

With basic charts, a universal eLSC can be defined as follows. A universal chart in the eLSCs consists of two basic charts: a *main-chart* (Mch, drawn within a solid rectangle) and a *pre-chart* (Pch). There are two possibilities of pre-charts: "*necessary pre-charts*" (drawn within a solid hexagon) and "*sufficient pre-charts*" (drawn within a hashed hexagon). These two pre-charts are interpreted as a necessary condition and a sufficient condition for a main-chart, respectively. Intuitively, a universal chart with a necessary pre-chart specifies all traces such that, if contains a segment which is admitted by the pre-chart, then it must also contain a continuation segment (directly following the first segment) which is admitted by the main chart. On the other hand, a universal chart with a sufficient pre-chart specifies all traces such that, if contains a segment which is admitted by the main-chart, then the segment must (directly) follows a prefix segment which is admitted by the pre-chart. Formally, the syntax of eLSCs is as follows.

Definition 3 (Syntax of universal eLSCs). *A universal eLSC is a tuple*

$$Uch \triangleq (Pch, Mch, Cate)$$

with $Cate \in \{Suff, Nec\}$ *denoting the category of the pre-chart. More specifically, the chart* $(Pch, Mch, Suff)$ *is with a sufficient pre-chart, and* (Pch, Mch, Nec) *is with a necessary pre-chart.*

We define PeLSCs by introducing *condition structure* and *assignment structure* into eLSCs.

An assignment structure is comprised of a function $v := s$ with v being a variable and s being the name of a parameter. The variable v is evaluated to the value of a parameter name p. The function is surrounded by a rectangle with a sandglass icon at the top right corner. A condition structure is comprised of a proposition prop surrounded by a rectangle. The proposition expresses the comparisons of parameter values. The notations for the assignment structure and the condition structure are shown in Fig. 2.

Fig. 2. Examples: an assignment structure (left) and a condition structure (right)

In a PeLSC, assignment structures and condition structures combine naturally with event occurrences. Intuitively, an assignment structure stores the value of a parameter carried by the combined event occurrence; and a condition structure expresses the features of a parameter carried by the combined event occurrence. Formally, the syntax of the two structures are given as follows.

Definition 4 (Syntax of assignment and condition structures). *Let Uch be an eLSC, $\mathfrak{o} \in \mathsf{EO}(Uch)$ an event occurrence of Uch, v a free variable, s a nominal, and prop a proposition. An assignment structure is defined as a tuple $\mathsf{assi} \triangleq (v, s, \mathfrak{o})$, where s represents the name of of a parameter. A condition structure is defined as a pair $\mathsf{cond} \triangleq \langle \mathsf{prop}, \mathfrak{o} \rangle$.*

With these structures, a PeLSC can be defined as follows.

Definition 5 (Syntax of PeLSCs). *A PeLSC is defined as a tuple $PU \triangleq (Uch, \mathsf{COND}, \mathsf{ASSI})$, where Uch is an eLSC, and COND and ASSI are the sets of condition structures and assignment structures appearing in Uch, respectively.*

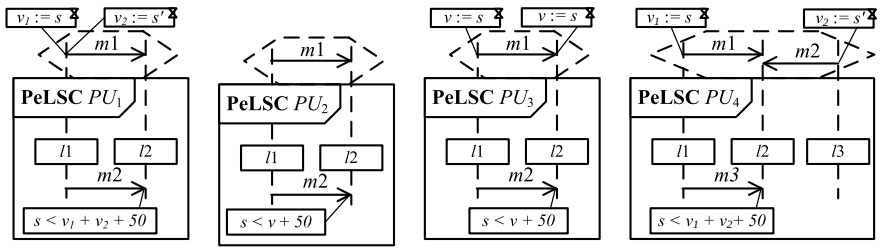
There are two possible forms of propositions in condition structures, one is with free variables (denoted by $\mathsf{prop}(s_1, ..., s_n, v_1, ..., v_m)$) and the other is without free variables (denoted by $\mathsf{prop}(s_1, ..., s_n)$). These two forms are used to express relative parameter values and absolute parameter values, respectively. We divide the set COND of a PeLSC into two subsets COND_{FV} and COND_{NFV}. The subset COND_{FV} is comprised of the set of condition structures with propositions of the form $\mathsf{prop}(s_1, ..., s_n, v_1, ..., v_m)$; and the subset COND_{NFV} is comprised of the set of condition structures with propositions of the form $\mathsf{prop}(s_1, ..., s_n)$. It holds that $(\mathsf{COND}_{FV} \cap \mathsf{COND}_{NFV}) = \emptyset$ and $(\mathsf{COND}_{FV} \cup \mathsf{COND}_{NFV}) = \mathsf{COND}$.

Given a parametrized event trace, the proposition in a condition structure $\langle \mathsf{prop}, \mathfrak{o} \rangle$ is evaluated to a boolean value, according to the parameter values carried by events:

- the nominals $s_1, ..., s_n$ in prop are replaced by the values of the parameters named by $s_1, ..., s_n$ carried by the event $lab(\mathfrak{o})$; and

– the variables $v_1, ..., v_m$ in prop are evaluated to the values from some event occurrences through the assignment structures $\mathsf{assi}(v_1, s_{x1}, \mathfrak{o}_{x1}), ...,$ assi $(v_m, s_{xm}, \mathfrak{o}_{xm})$.

In this paper, we require our PeLSCCs to satisfy an additional *non-ambiguity* assumption. We say a PeLSC is non-ambiguity, if for any condition structure with a proposition of the form $\mathsf{prop}(s_1, ..., s_n, v_1, ..., v_m)$, all free variable $v_1, ..., v_m$ are evaluated to a certain value from a unique event occurrence. More formally, a PeLSC $PU \triangleq (Uch, \mathsf{COND}, \mathsf{ASSI})$ is non-ambiguity if and only if for any condition structure $\mathsf{cond} = (\mathsf{prop}(s_1, ..., s_n, v_1, ..., v_m), \mathfrak{o})$ in the set COND_{FV} it holds that for any $v_{xi} \in \{v_1, ..., v_m\}$ there exists an assignment structure $(v_{xi}, s_{xi}, \mathfrak{o}') \in \mathsf{ASSI}$ with $\mathfrak{o}' \prec \mathfrak{o}$. With the non-ambiguity assumption, each proposition is able to be evaluated to a certain boolean value (*true* or *false*) over a deterministic parametrized event trace. To understand this assumption, consider the PeLSCs in Fig. 3. For the chart PU_2, the variable v has no value since there is no assignment structure to store it. For the chart PU_3, the variable v has two values stored by two assignment structures. Therefore, the condition structure cannot be evaluated to a certain boolean value for both of the two charts. For the chart PU_4, the values of variables v_1 and v_2 are from different event occurrences.



(a) non-ambiguity (b) ambiguity (c) ambiguity (d) ambiguity

Fig. 3. Examples: non-ambiguity assumptions of PeLSCs

The non-ambiguity assumption is a strong assumption. For instance, the variables v_1 and v_2 in the chart PU_4 have certain values. However, the order of the two events $!m1$ and $!m2$, which are combined with the two assignment structures of v_1 and v_2, are uncertain. Since our monitors are generated by translating PeLSCs into **HL**, the size of the monitor is increased by expressing all possible executions of the pre-chart in the resulting formula. This will reduce the monitoring efficiency.

3.2 The Trace-based Semantics of PeLSCs

A PeLSC $(Uch, \mathsf{COND}, \mathsf{ASSI})$ defines a parametrized language (a set of parametrized event traces) that is an extension of the propositional language (i.e., a set of event traces) defined by Uch. Intuitively, the parametrized language is comprised of all parametrized event traces such that the orders of events meets the partial order induced by Uch, and all propositions in COND are evaluated to *true* with the values from the parameters carried by the events. A parametrized event trace τ is *admitted* by a PeLSC PU if and only if τ is in the parametrized language defined by PU.

A set of sequences of event occurrences is defined by a basic chart C as follows:

$EOcc(C) \triangleq \{(\mathfrak{o}[x1], \mathfrak{o}[x2], ..., \mathfrak{o}[xn]) | \{\mathfrak{o}[x1], \mathfrak{o}[x2], ..., \mathfrak{o}[xn]\} = \mathsf{EO}(C); n = |\mathsf{EO}(C)|$; and for all $\mathfrak{o}[xi], \mathfrak{o}[xj] \in \mathsf{EO}(C)$, if $\mathfrak{o}[xi] \prec \mathfrak{o}[xj]$, then $xi < xj\}$.

For languages $\mathcal{L}$ and $\mathcal{L}'$, let $(\mathcal{L} \circ \mathcal{L}')$ be the concatenation of $\mathcal{L}$ and $\mathcal{L}'$ (i.e., $(\mathcal{L} \circ \mathcal{L}') \triangleq \{(\sigma\sigma') \mid \sigma \in \mathcal{L} \ \textit{and} \ \sigma' \in \mathcal{L}'\}$); and $\overline{\mathcal{L}}$ be the complement of $\mathcal{L}$ (i.e., for any $\sigma \in \Sigma^*$, it holds that $\sigma \in \overline{\mathcal{L}}$ iff $\sigma \notin \mathcal{L}$). The language of event occurrences for eLSCs is given as follows.

Definition 6 (Semantics of universal eLSCs). *The language of event occurrences defined by an eLSC* $Uch \triangleq (Pch, Mch, Cate)$ *is as follows*

- $EOcc(Uch) \triangleq \overline{\overline{EOcc(Pch)} \circ \overline{EOcc(Mch)}}$, *if* $Cate = Suff$;
- $EOcc(Uch) \triangleq \overline{\overline{EOcc(Pch)} \circ EOcc(Mch)}$, *if* $Cate = Nec$.

The evaluations of the propositions are formally defined as follows.

Given a set $\mathcal{P} = \{\langle s_1, d_1\rangle, ..., \langle s_n, d_m\rangle\}$ of parameters and a proposition $\mathsf{prop}(s_{y1}, ..., s_{yn})$, we define the evaluation of $\mathsf{prop}(s_{y1}, ..., s_{yn})$ over $\mathcal{P}$ (denoted with $[\![\mathcal{P} \vdash \mathsf{prop}(s_{y1}, ..., s_{yn})]\!]$) as follows.

$$[\![\mathcal{P} \vdash \mathsf{prop}(s_{y1}, ..., s_{yn})]\!] = \begin{cases} true & \textit{if } \mathsf{prop}(d_{y1}, ..., d_{yn}) \textit{ is satisfied with} \\ & \{\langle s_{y1}, d_{y1}\rangle, ..., \langle s_{yn}, d_{ym}\rangle\} \subseteq \mathcal{P} \\ false & \textit{otherwise} \end{cases}$$

For the propositions of the form $\mathsf{prop}(s_{x1}, ..., s_{xn}, v_{y1}, ..., v_{ym})$ in PU, the values of the variables $v_{y1}, ..., v_{ym}$ are from the assignment structures $\mathsf{ASSI}(v_{y1}, ..., v_{ym}) \triangleq \{\mathsf{assi}(v_{y1}), ..., \mathsf{assi}(v_{ym})\}$, where $\mathsf{assi}(v_{yi}) \triangleq (v_{yi}, s_{yi}, \mathfrak{o})$. By $\mathsf{ac}(s_{x1}, ..., s_{xn}, v_{y1}, ..., v_{ym})$ we denote the pair $(\mathsf{prop}(s_{x1}, ..., s_{xn}, v_{y1}, ..., v_{ym}), \mathsf{ASSI}(v_{y1}, ..., v_{ym}))$. Given a pair $\langle \mathcal{P}, \mathcal{P}'\rangle$ of two sets of parameters, the evaluation of the pair $\mathsf{ac}(s_{x1}, ..., s_{xn}, v_{y1}, ..., v_{ym})$ over $\langle \mathcal{P}, \mathcal{P}'\rangle$ is defined as follows.

- $[\![\langle \mathcal{P}, \mathcal{P}'\rangle \vdash \mathsf{ac}(s_{x1}, ..., s_{xn}, v_{y1}, ..., v_{ym})]\!] = true$, if $\mathsf{prop}(d_{x1}, ..., d_{xn}, d_{y1}, ..., d_{ym})$ is satisfied with $\{(s_{x1}, d_{x1}), ..., (s_{xn}, d_{xn})\} \subseteq \mathcal{P}'$ and $\{(s_{y1}, d_{y1}), ..., (s_{yn}, d_{yn})\} \subseteq \mathcal{P}$;
- $[\![\langle \mathcal{P}, \mathcal{P}'\rangle \vdash \mathsf{ac}(s_{x1}, ..., s_{xn}, v_{y1}, ..., v_{ym})]\!] = false$, otherwise.

A PeLSC PU defines all parametrized event traces (σ, ρ) such that

– $\sigma = (lab(\mathfrak{o}[1]), ..., lab(\mathfrak{o}[n]))$ *with* $(\mathfrak{o}[1], ..., \mathfrak{o}[n]) \in EOcc(Uch)$;
– for all $\langle \mathsf{prop}, \mathfrak{o} \rangle \in \mathsf{COND}_{NFV}$, if there exists $zi \in [1, n]$ with $\mathfrak{o}[zi] = \mathfrak{o}$, then $[\![\rho[zi] \vdash \mathsf{prop}]\!] = true$; and
– for all $\langle \mathsf{prop}(s_{x1}, ..., s_{xn}, v_{y1}, ..., v_{ym}), \mathfrak{o} \rangle \in \mathsf{COND}_{FV}$ and $(v_{yi}, s_{yi}, \mathfrak{o}') \in \mathsf{ASSI}(v_{y1}, ..., v_{ym})$, if there exists $zj, zk \in [1, n]$ such that $\mathfrak{o}[zj] = \mathfrak{o}'$ and $\mathfrak{o}[zk] = \mathfrak{o}$, then $[\![\langle \rho[zj], \rho[zk] \rangle \vdash \mathsf{prop}]\!] = true$.

By $PTra(PU)$ we denote the set of parametrized event traces defined by PB. Let $\mathcal{E}(Uch)$ be the set of events appearing in an eLSC. We call each $\epsilon_{Uch} \in (\Sigma \backslash \mathcal{E}(Uch))$ a *stutter event*, and $\mathcal{P}_\epsilon \in 2^{\mathcal{N}\mathcal{D}}$ an arbitrary set of *stutter parameters*.

Definition 7 (Trace based semantics for PeLSCs). *The parametrized language defined by a PeLSC PU is*

$$\mathcal{PL}(PU) \triangleq \{(\epsilon^*, e_1, \epsilon^*, ..., e_n, \epsilon^*), (\mathcal{P}_\epsilon^*, \mathcal{P}_1, \mathcal{P}_\epsilon^* ..., \mathcal{P}_n, \mathcal{P}_\epsilon^*)\},$$

where $((e_1, ..., e_n), (\mathcal{P}_1, ..., \mathcal{P}_n)) \in PTra(PU)$*, and* $|(\epsilon^*, e_1, \epsilon^*, ..., e_n, \epsilon^*)| = |(\mathcal{P}_\epsilon^*, \mathcal{P}_1, \mathcal{P}_\epsilon^* ..., \mathcal{P}_n, \mathcal{P}_\epsilon^*)|$*, and* ϵ^* *and* $\mathcal{P}_\epsilon^*$ *are finite (possible empty) sequences of stutter events and stutter parameters, respectively.*

4 A Translation of PeLSCs into HL formulae

In this subsection, we present a translation of PeLSCs into a subclass of the hybrid logic (**HL**) formulae. Whether an observation is admitted by a PeLSC can then be checked with the resulting formula.

The syntax and semantics of **HL** are given as follows.

Definition 8 (Syntax of HL). *Given the finite set AP of atomic propositions, a set V of variables, the set* $\mathbb{Z}$ *of integers, and a set* $\mathcal{N}$ *of nominals, the terms* π *and formulae* φ *of* **HL** *are inductively formed according to the following grammar, where* $x \in V$*,* $p \in AP$*,* $s \in \mathcal{N}$*,* $r \in \mathbb{Z}$ *and* $\sim \in \{<, =\}$*:*

$\pi ::= x + r \mid r$
$\varphi ::= \bot \mid p \mid (\varphi_1 \Rightarrow \varphi_2) \mid (\varphi_1 \; \mathcal{U} \; \varphi_2) \mid (\pi_1 \sim \pi_2) \mid x \downarrow s.\varphi.$

Intuitively, an **HL** formula $x \downarrow s.\varphi(x)$ is satisfied by a parametrized event trace $\tau \triangleq (\sigma, \rho)$ if and only if $\varphi(d)$ is satisfied by σ with $(s, d) \in \rho[1]$. For instance, let t and id be parameters representing time stamps and clients' ID, respectively, a formula

$$\Box x \downarrow t.y \downarrow id.(login \Rightarrow \Diamond x' \downarrow t.y' \downarrow id.(logout \wedge (y' = y) \wedge (x' < 200 + x)))$$

expresses the property **Pro**.

Assume that $\mathcal{E}$ is a function $\mathcal{E}$: $V \to \mathbb{Z}$ for assigning free variables in the domain of integers $\mathbb{N}_{\geq 0}$ such that $\mathcal{E}(x + d) = \mathcal{E}(x) + d$ and $\mathcal{E}(d) = d$. Given a variable x and a natural number d, we denote $\mathcal{E}[x := d]$ for the evaluation $\mathcal{E}'$ such that $\mathcal{E}'(x) = d$, and $\mathcal{E}'(y) = \mathcal{E}(y)$ for all $y \in V \backslash \{x\}$. The **HL** is defined on parametrized event traces as follows.

Definition 9 (Trace-based Semantics for HL).

Let $\tau \triangleq (\sigma, \rho)$ *be a parametrized event trace with* $\sigma \triangleq (e[1], e[2], ...)$ *being an event trace and* $\rho \triangleq (\mathcal{P}[1], \mathcal{P}[2]...,)$ *being a parameter trace. Let* $i \in \mathbb{Z}_{>0}$ *be a position,* p *a proposition,* s *a nominal,* d *a value in the domain of parameters, and* φ_1 *and* φ_2 *any* **HL** *formulae. The satisfaction relation* $(\tau, i, \mathcal{E}) \models \varphi$ *is defined inductively as follows:*

$(\tau, i, \mathcal{E}) \not\models \bot$;

$(\tau, i, \mathcal{E}) \models p$ *iff* $p \in e[i]$;

$(\tau, i, \mathcal{E}) \models (\varphi_1 \Rightarrow \varphi_2)$ *iff* $(\tau, i, \mathcal{E}) \models \varphi_1$ *implies* $(\tau, i, \mathcal{E}) \models \varphi_2$;

$(\tau, i, \mathcal{E}) \models (\varphi_1 \ \mathcal{U} \ \varphi_2)$ *iff there exists* $j > i$ *with* $(\tau, j, \mathcal{E}) \models \varphi_2$ *and for all* $i < j' < j$ *it holds that* $(\tau, j', \mathcal{E}) \models \varphi_1$;

$(\tau, i, \mathcal{E}) \models \pi_1 \sim \pi_2$ *iff* $\mathcal{E}(\pi_1) \sim \mathcal{E}(\pi_2)$;

$(\tau, i, \mathcal{E}) \models x \downarrow s.\varphi$ *iff* $(\tau, i, \mathcal{E}[x := d]) \models \varphi$, *where* $(s, d) \in \mathcal{P}[i]$.

As usual, $\tau \models \varphi$ iff $(\tau, 1, \mathcal{E}) \models \varphi$.

We now show how to translate a PeLSC into an HL formula to check whether a parametrized trace is admitted. The translation is developed on basis of the translation form an eLSC Uch into an LTL formulae $\varphi(Uch)$ as shown in [14]. Here we concern the translation of the introduced assignment and condition structures for PeLSCs.

A PeLSC is comprised of a universal eLSC Uch, a set COND of condition structures and a set ASSI of assignment structures. Propositions appearing in a PeLSC are specified by the comparisons of terms, i.e., $\pi_1 \sim \pi_2$. According to the subset COND_{NFV} and COND_{FV}, the following formulae are defined.

- Let e be an even in Uch combined with a constraint structure $\mathsf{cond} = (\mathsf{prop}, \mathfrak{o}$ from the set COND_{NFV} with $\mathsf{prop}(s_1, ..., s_n)$ and $lab(\mathfrak{o}) = e$. The condition structure is translated into an **HL** formula

$$\mathfrak{d}(\mathsf{cond}) \triangleq \Box\ (x_1 \downarrow s_1. \cdots x_n \downarrow s_n.(e \Rightarrow \mathsf{prop}(x_1, ..., x_n))).$$

  The formula specifies that whenever the event e occurs, then the proposition must be evaluated to *true* with values of the parameters named by $s_1, ..., s_n$ carried by e. I.e., if the event occurs at a position $z1$ of the trace, the proposition $\mathsf{prop}(d_1, ..., d_n)$ is true with $\{(s_1, d_1), ..., (s_n, d_n)\} \subseteq \rho[z1]$.
- For a condition structure $\mathsf{cond} \in \mathsf{COND}_{FV}$, there is a tuple $\mathsf{ac}(\mathsf{cond}) \triangleq (\mathsf{prop}(s_{zy1}, ..., s_{zyn}, v_{zx1}, ..., v_{zxm}), \mathsf{o}, \mathsf{ASSI}(v_{zx1}, .., v_{zxm}))$ with $\mathsf{ASSI}(v_{zx1}, .., v_{zxm}) \triangleq \{\mathsf{assi}(v_{zx1}), ..., \mathsf{assi}(v_{zxm})\}$ and $\mathsf{assi}(v_{zxi}) \triangleq (v_{zxi}, s_{zxi}, \mathfrak{o}')$ for any $1 \leq i \leq m$. Let $e = lab(\mathfrak{o})$ and $e' = lab(\mathfrak{o}')$ be the events of the two event occurrences, the condition structure is translated into an **HL** formula

$$\mathfrak{d}(\mathsf{cond}) \triangleq \Box((e' \wedge \Diamond e) \Rightarrow (x_1 \downarrow v_1. \cdots x_m \downarrow v_m.(e' \wedge \Diamond y_1 \downarrow s_1. \cdots y_n \downarrow s_n.(e \wedge (\mathsf{prop}(y_1, ..., y_n, x_1, ..., x_m)))))).$$

  This formula expresses that if both of the events, combined with the assignment structures and with the condition structure, occurs, then the proposition must be evaluated to *true* with the values of the parameters carried by the two events. I.e., if e' and e occur at positions $z1$ and $z2$ of the trace,

respectively, the proposition $\mathsf{prop}(d_{zy1}, ..., d_{zyn}, d_{zx1}, ..., d_{zxm})$ is true with $\{(s_{zy1}, d_{zy1}), ..., (s_{zyn}, d_{zyn})\} \subseteq \rho[z2]$ and $\{(s_{zx1}, d_{zx1}), ..., (s_{zxm}, d_{zxm})\} \subseteq \rho[z1]$.

From a PeLSC $PU \triangleq (Uch, \mathsf{COND}, \mathsf{ASSI})$ we define an **HL** formula

$$\varphi(PU) \triangleq \left(\varphi(Uch) \wedge \bigwedge_{\mathsf{cond} \in \mathsf{COND}_{NFV}} \mathfrak{d}(\mathsf{cond}) \wedge \bigwedge_{\mathsf{cond} \in \mathsf{COND}_{FV}} \mathfrak{d}(\mathsf{cond}) \right).$$

The formula expresses that, a parametrized event trace $\tau = (\sigma, \rho)$ satisfies the formula $\varphi(PU)$ if and only if σ is in the language defined by Uch, and the parameters carried by the events in τ meet the specification of the assignment structures and the condition structures. For any parametrized event trace τ, it holds that τ is admitted by PU iff $\tau \models \varphi(PU)$.

A rewriting algorithm for **HL** can be developed directly upon the semantics of the logic. Then a PeLSC property can be checked by a monitor by implementing the algorithm in some rewriting environment, e.g, Maude [27].

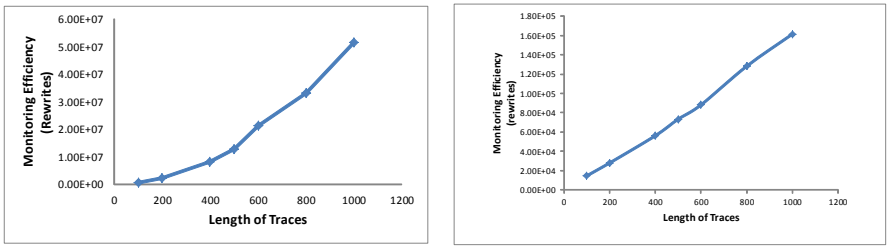
Theorem 1. *The complexity of the word-problem of* **HL** *is linear with respect to the size of input traces.*

Proof. Given an **HL** formula $\varphi = x \downarrow s.\psi(x)$ and a parametrized event trace $\tau \triangleq (\sigma, \rho)$. The trace τ satisfies φ if and only if $\sigma \models \psi(d)$ with $(s, d) \in \rho[1]$. Since d is a certain integer value, the sub-formulae of the comparisons of terms in φ is able to be directly evaluated a boolean value $true$ or $false$. Therefore, the process of checking an **HL** formula over a parametrized event trace τ is essentially the same with the process of checking an LTL formula over an event trace σ. Since the complexity of checking whether or not a trace σ satisfies an LTL formula is linear with respect to the size of the trace σ, the complexity of the word problem of **HL** is linear with respect to the length of the input parametrized event traces.

Corollary 1. *The complexity of PeLSCs is linear with respect to the size of traces.*

According to the corollary, the language of PeLSCs is feasible for runtime verification implementations, especially for on-line monitoring.

We implement the algorithms in Maude [17], which provides a formula rewriting environment for monitoring. The implementation is valuated on several benchmarks. The monitoring efficiency for the property **P3** is shown in Fig. 4. The property **P3** is comprised of an eLSC and two condition structures with assignments (i.e., with free variables). Fig. 4(a) shows the monitoring efficiency for the condition structures, and Fig. 4(b) shows the monitoring efficiency for the eLSC. In this monitoring implementation, the most rewrites are spent on monitoring the condition structures with free variables.



(a) Parametric requirements (b) Temporal requirements

Fig. 4. Monitoring efficiency for **P3**

5 Conclusion

In this paper, we defined PeLSCs for parametric properties by introducing assignment and condition structures into LSCs. With these structures, PeLSC can be interpreted over parametrized event traces. The language can than intuitively express requirement of data (e.g., values of time or other variables) carried by events. We developed translation from PeLSCs into **HL** for monitoring. We prove that the complexity of the word problem of PeLSCs is linear if propositions in the condition structures only express comparisons of terms.

There are several interesting topics for future work. Firstly, in this paper, we only concerned comparisons of terms in the condition structures. It is interesting to find out whether or not the PeLSC is still feasible for monitoring if the expressiveness of conditions is extended by, e.g., introducing quantifiers $\forall$ and $\exists$. Secondly, since the sizes of resulting formulae are often large, translating PeLSC into **HL** formulae is not an efficient way for monitoring. Therefore, we are currently developing a more efficient implementation, which can check PeLSC specifications directly. Last but not least, the synthesis problem of PeLSC based monitors is left open in this paper. As PeLSCs have features of the first order logic, the existing LSC synthesising techniques cannot handle this problem.

References

1. Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Ondřej Lhoták, Oege De Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. Adding Trace Matching with Free Variables to AspectJ. In *ACM SIGPLAN Notices*, volume 40, pages 345–364. ACM, 2005.
2. Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of Message Sequence Charts. *Software Engineering, IEEE Transactions on*, 29(7):623–633, 2003.
3. Howard Barringer, Ylies Falcone, Klaus Havelund, Giles Reger, and David Rydeheard. Quantified Event Automata: Towards Expressive and Efficient Runtime Monitors. In *FM 2012: Formal Methods*, pages 68–84. Springer, 2012.
4. Howard Barringer, Michael Fisher, Dov Gabbay, Graham Gough, and Richard Owens. MetateM: A Framework for Programming in Temporal Logic. In J.W.

de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Stepwise Refinement of Distributed Systems Models, Formalisms, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 94–129. Springer Berlin Heidelberg, 1990.

5. Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. Rule-based Runtime Verification. In *Verification, Model Checking, and Abstract Interpretation*, pages 44–57. Springer, 2004.
6. Howard Barringer and Klaus Havelund. TraceContract: A Scala DSL for Trace Analysis. In Michael Butler and Wolfram Schulte, editors, *FM 2011: Formal Methods*, volume 6664 of *Lecture Notes in Computer Science*, pages 57–72. Springer Berlin Heidelberg, 2011.
7. Howard Barringer, David Rydeheard, and Klaus Havelund. Rule Systems for Run-time Monitoring: from Eagle to RuleR. *Journal of Logic and Computation*, 20(3):675–706, 2010.
8. David Basin, Germano Caronni, Sarah Ereth, Matúš Harvan, Felix Klaedtke, and Heiko Mantel. Scalable Offline Monitoring. In Borzoo Bonakdarpour and ScottA. Smolka, editors, *Runtime Verification*, volume 8734 of *Lecture Notes in Computer Science*, pages 31–47. Springer International Publishing, 2014.
9. David Basin, Felix Klaedtke, and Samuel Müller. Policy Monitoring in First-order Temporal Logic. In *Computer Aided Verification*, pages 1–18. Springer, 2010.
10. David Basin, Felix Klaedtke, and Eugen Zălinescu. Algorithms for Monitoring Real-time Properties. In *Runtime Verification*, pages 260–275. Springer, 2012.
11. Andreas Bauer, Jan-Christoph Küster, and Gil Vegliach. From Propositional to First-order Monitoring. In *Runtime Verification*, pages 59–75. Springer, 2013.
12. Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime Verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):14, 2011.
13. Patrick Blackburn and Jerry Seligman. Hybrid Languages. *Journal of Logic, Language and Information*, 4(3):251–272, 1995.
14. Ming Chai and Bernd-Holger Schlingloff. Monitoring Systems with Extended Live Sequence Charts. In *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, pages 48–63, 2014.
15. Ming Chai and Holger Schlingloff. A Rewriting Based Monitoring Algorithm for TPTL. In *CS&P 2013*, pages 61–72. Citeseer, 2013.
16. Feng Chen and Grigore Roşu. Mop: an Efficient and Generic Runtime Verification Framework. In *ACM SIGPLAN Notices*, volume 42, pages 569–588. ACM, 2007.
17. Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martı-Oliet, José Meseguer, and Carolyn Talcott. Maude Manual (version 2.6). *University of Illinois, Urbana-Champaign*, 1(3):4–6, 2011.
18. Werner Damm and David Harel. LSCs: Breathing Life into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001.
19. Charles L Forgy. Rete: A Fast Algorithm for the Many Pattern/many Object Pattern Match Problem. *Artificial intelligence*, 19(1):17–37, 1982.
20. Sylvain Halle and Roger Villemaire. Runtime Monitoring of Message-based Workflows with Data. In *Enterprise Distributed Object Computing Conference, 2008. EDOC'08. 12th International IEEE*, pages 63–72. IEEE, 2008.
21. David Harel and Rami Marelly. Playing with Time: On the Specification and Execution of Time-enriched LSCs. In *Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings. 10th IEEE International Symposium on*, pages 193–202. IEEE, 2002.

22. Klaus Havelund. Monitoring with Data Automata. In *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*, pages 254–273. Springer, 2014.
23. Klaus Havelund. Monitoring with Data Automata. In *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications*, pages 254–273. Springer, 2014.
24. Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G Griswold. An Overview of AspectJ. In *ECOOP 2001Object-Oriented Programming*, pages 327–354. Springer, 2001.
25. Patrick ONeil Meredith, Dongyun Jin, Dennis Griffith, Feng Chen, and Grigore Roşu. An Overview of the MOP Runtime Verification Framework. *International Journal on Software Tools for Technology Transfer*, 14(3):249–289, 2012.
26. Stephan Merz. Decidability and Incompleteness Results for First-order Temporal Logics of Linear Time. *Journal of Applied Non-Classical Logics*, 2(2):139–156, 1992.
27. Peter Csaba Olveczky. Real-time Maude 2.3 Manual. *Research report*, 2004.
28. Volker Stolz. Temporal Assertions with Parametrized Propositions. *Journal of Logic and Computation*, 20(3):743–757, 2010.