Monitoring with Parametrized Extended Life Sequence Charts*

Ming Chai\textsuperscript{1} and Bernd-Holger Schlingloff\textsuperscript{2}

\textsuperscript{1} Humboldt Universität zu Berlin  \\
\textsuperscript{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 life 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 (LTLFO). 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.
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 (id = x) and (time = y), a logout event with (id = x') and (time = y') should occur afterwards, where (x' = x) and (y' ≤ (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 ↓ 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 subclass 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 $\mu$-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$^O$ includes both first-order and temporal connectives [26]. For monitoring LTL$^O$, an algorithm using a spawning automaton has been developed [11]. However, the word problem of LTL$^O$ 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. Similar 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 $\Sigma \triangleq \{e_1, e_2, \ldots, e_n\}$ be a finite alphabet of events, $\mathcal{N} \triangleq \{s_1, s_2, \ldots\}$ be a countable set of nominals and $\mathcal{D} \triangleq \{d_1, d_2, \ldots\}$ any domain (e.g., integers, strings, or reals). A **parameter** is a pair $p \triangleq (s, d)$ 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 \( \Sigma \) of events, a set \( N \) of nominals and a domain \( D \), a parametrized event is a pair \( w \triangleq (e, P) \), where \( e \in \Sigma \) is an event and \( P \in 2^{N \times D} \) is a set of parameters.

Given a parametrized event \( w \) with \( P \triangleq \{ (s_1, d_1), \ldots, (s_m, d_m) \} \), we define \( \text{Evet}(w) \triangleq e \), \( \text{Para}(w) \triangleq P \) and \( \text{Nam}(w) \triangleq \{ s_1, \ldots, s_m \} \). A parametrized event \( \langle e, P \rangle \) is deterministic if each parameter name in \( P \) is unique, i.e., for all \( p, p' \in P \) 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 \( N \) and \( D \), a parameter trace \( \rho \triangleq (P_1, P_2, \ldots, P_n) \) over \( N \times D \) is a finite sequence of sets of parameters, i.e., an element of \( (2^{N \times D})^* \). Given \( \Sigma, N \) and \( D \), a parametrized event trace \( \tau \triangleq (\sigma, \rho) \) is a pair of a finite event trace \( \sigma \) and a parameter trace \( \rho \) with the same length, i.e., \( \sigma \in \Sigma^* \) and \( \rho \in (2^{N \times D})^* \) and \( |\sigma| = |\rho| \).

By \( \tau[i] \triangleq \langle \sigma[i], \rho[i] \rangle \) we denote the \( i^{th} \) parametrized event of \( \tau \), where \( \sigma[i] \) and \( \rho[i] \) are the \( i^{th} \) element of \( \sigma \) and \( \rho \), 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 \( l \triangleq (e_1, e_2, \ldots, e_n) \). A basic chart \( C \) is an \( n \)-tuple of lifelines \( (l_1, \ldots, l_n) \) with \( l_i = (e_{i1}, \ldots, e_{im}) \). We say that an event \( e \) occurs at the location \((i, j)\) in chart \( C \) if \( e = e_{ij} \). An event occurrence \( o \triangleq (e, i, j) \) is a tuple consisting of an event \( e \) and the location of \( e \). We define \( \text{loc}(o) \triangleq (i, j) \) as the location of an event occurrence \( o \), and \( \text{lab}(o) \triangleq e \) is the event of \( o \). We denote the set of event occurrences appearing in \( C \) with \( \text{EO}(C) \). 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 \( \text{mat}(m!, i, j) \triangleq (m?, i, j) \) 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 \( \text{EO}(C) \) is formalized as follows.

1. for any \( 1 \leq x < |l_x| \) with \( l_x \) being a lifeline in \( C \), it holds that \( (e, x, x) \prec (e', x, (x + 1)) \);
2. for any \( o \in S \), it holds that \( o \prec \text{mat}(o) \); 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 \( o_1, o_2 \in S \) with \( o_1 \prec o_2 \) such that \( \text{lab}(o_1) = \text{lab}(o_2) \) and \( \text{mat}(o_1) \not\prec \text{mat}(o_2) \).

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 \( (Mch, \text{drawn within a solid rectangle}) \) 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

\[
\text{Uch} \triangleq (Pch, Mch, \text{Cate})
\]

with \( \text{Cate} \in \{\text{Suff, Nec}\} \) denoting the category of the pre-chart. More specifically, the chart \( (Pch, Mch, \text{Suff}) \) is with a sufficient pre-chart, and \( (Pch, Mch, \text{Nec}) \) 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 \( \text{prop} \) 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 \( \text{Uch} \) be an \( eLSC \), \( o \in \text{EO}(\text{Uch}) \) an event occurrence of \( \text{Uch} \), \( v \) a free variable, \( s \) a nominal, and \( \text{prop} \) a proposition. An assignment structure is defined as a tuple \( \text{assi} \defeq (v, s, o) \), where \( s \) represents the name of a parameter. A condition structure is defined as a pair \( \text{cond} \defeq (\text{prop}, o) \).

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

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

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

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

- the nominals \( s_1, ..., s_n \) in \( \text{prop} \) are replaced by the values of the parameters named by \( s_1, ..., s_n \) carried by the event \( \text{lab}(o) \); and
the variables $v_1, ..., v_m$ in $\text{prop}$ are evaluated to the values from some event occurrences through the assignment structures

$$\text{assi}(v_1, s_{x1}, o_{x1}), ..., \text{assi}(v_m, s_{xm}, o_{xm})$$

In this paper, we require our PeLSCCs to satisfy an additional \textit{non-ambiguity} assumption. We say a PeLSC is non-ambiguity, if for any condition structure with a proposition of the form $\text{prop}(s_1, ..., s_n, v_1, ..., v_m)$, all free variable $v_1, ..., v_m$ are evaluated to a certain value from a unique event occurrence. More formally, a PeLSC $PU \equiv (Uch, \text{COND}, \text{ASSI})$ is non-ambiguity if and only if for any condition structure $\text{cond} = (\text{prop}(s_1, ..., s_n, v_1, ..., v_m), o)$ in the set $\text{COND}_{FV}$ it holds that for any $v_{xi} \in \{v_1, ..., v_m\}$ there exists an assignment structure $(v_{xi}, s_{xi}, o') \in \text{ASSI}$ with $o' \prec o$. 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.

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 $\text{HL}$, 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, COND, 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 \( \tau \) is admitted by a PeLSC \( PU \) if and only if \( \tau \) is in the parametrized language defined by \( PU \).

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

\[
EOcc(C) \equiv \{ ([x_1], [x_2], ..., [x_n]) | [x_1], [x_2], ..., [x_n] \} = EO(C); n = |EO(C)|; \text{ and for all } [x_i], [x_j] \in EO(C), \text{ if } [x_i] < [x_j], \text{ then } x_i < x_j.
\]

For languages \( \mathcal{L} \) and \( \mathcal{L}' \), let \( (\mathcal{L} \circ \mathcal{L}') \) be the concatenation of \( \mathcal{L} \) and \( \mathcal{L}' \) (i.e., \( (\mathcal{L} \circ \mathcal{L}') \equiv \{ ([\sigma]) | \sigma \in \mathcal{L} \text{ and } \sigma' \in \mathcal{L}' \} \)); 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 \( \equiv (Pch, Mch, Cate) \) is as follows

\[
- EOCC(Uch) \equiv EOCC(Pch) \circ EOCC(Mch), \text{ if Cate = Suff};
- EOCC(Uch) \equiv EOCC(Pch) \circ EOCC(Mch), \text{ if Cate = Nec}.
\]

The evaluations of the propositions are formally defined as follows.

Given a set \( \mathcal{P} = \{ (s_1, d_1), ..., (s_n, d_m) \} \) of parameters and a proposition \( \text{prop}(s_1, ..., s_m) \), we define the evaluation of \( \text{prop}(s_1, ..., s_m) \) over \( \mathcal{P} \) (denoted with \( \mathcal{P} \vdash \text{prop}(s_1, ..., s_m) \)) as follows.

\[
[\mathcal{P} \vdash \text{prop}(s_1, ..., s_m)] = \begin{cases} 
\text{true} & \text{if } \text{prop}(d_1, ..., d_n) \text{ is satisfied with } (s_1, d_1), ..., (s_m, d_m) \subseteq \mathcal{P} \\
\text{false} & \text{otherwise}
\end{cases}
\]

For the propositions of the form \( \text{prop}(s_1, ..., s_n, v_1, ..., v_m) \) in \( PU \), the values of the variables \( v_1, ..., v_m \) are from the assignment structures

\[
\text{ASSI}(v_1, ..., v_m) \equiv \{ \text{assi}(v_1), ..., \text{assi}(v_m) \}, \text{ where } \text{assi}(v_1) \equiv (v_1, s_1, o). \]

By \( \text{ac}(s_1, ..., s_n, v_1, ..., v_m) \) we denote the pair \( (\text{prop}(s_1, ..., s_n, v_1, ..., v_m), \text{ASSI}(v_1, ..., v_m)) \).

Given a pair \( \langle \mathcal{P}, \mathcal{P}' \rangle \) of two sets of parameters, the evaluation of the pair \( \text{ac}(s_1, ..., s_n, v_1, ..., v_m) \) over \( \langle \mathcal{P}, \mathcal{P}' \rangle \) is defined as follows.

\[
- [[\mathcal{P}, \mathcal{P}'] \vdash \text{ac}(s_1, ..., s_n, v_1, ..., v_m)] = \text{true}, \text{ if } \text{prop}(d_1, ..., d_n, d_1, ..., d_m) \text{ is satisfied with } (s_1, d_1), ..., (s_n, d_n) \subseteq \mathcal{P} \text{ and } (s_1, d_1), ..., (s_m, d_m) \subseteq \mathcal{P}'
\]

A PeLSC \( PU \) defines all parametrized event traces \( (\sigma, \rho) \) such that
Definition 7 (Trace based semantics for PeLSCs). The parametrized language defined by a PeLSC \((\Sigma, \mathcal{E}(Uch))\) a stutter event, and \(\mathcal{P}_e \in 2^{ND}\) an arbitrary set of stutter parameters.

**Definition 7 (Trace based semantics for PeLSCs).** The parametrized language defined by a PeLSC \(\mathcal{P}L(\mathcal{P}U)\) is

\[
\mathcal{P}L(\mathcal{P}U) \triangleq \{(\epsilon^*, \epsilon_1, ..., \epsilon_n, \epsilon^*), (\mathcal{P}_{e^*}, \mathcal{P}_{1^*}, ..., \mathcal{P}_{n^*})\},
\]

where \((\epsilon_1, ..., \epsilon_n), (\mathcal{P}_1, ..., \mathcal{P}_n)\) \(\in \) \(PTra(\mathcal{P}U)\), and

\[
[(\epsilon^*, \epsilon_1, ..., \epsilon_n, \epsilon^*)] = [(\mathcal{P}_{e^*}, \mathcal{P}_{1^*}, ..., \mathcal{P}_{n^*})], \text{ and } \epsilon^* \text{ and } \mathcal{P}_{e^*} \text{ are finite (possibly empty) sequences of stutter events and stutter parameters, respectively.}
\]

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 \(\pi\) and formulae \(\varphi\) of HL are inductively formed according to the following grammar, where \(x \in V, p \in AP, s \in N, r \in \mathbb{Z}\) and \(\sim \in \{<, =\}:

\[
\pi ::= x + r | r
\]

\[
\varphi ::= \bot | p | (\varphi_1 \Rightarrow \varphi_2) | (\varphi_1 \bigcup \varphi_2) | (\pi_1 \sim \pi_2) | x \downarrow s.\varphi.
\]

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 \(\sigma\) 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 \(\mathcal{E}: V \rightarrow \mathbb{Z}\) 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 \(y \in \mathcal{V}\backslash \{x\}\). 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 \( \rho \triangleq (P[1], P[2], ...) \) being a parameter trace. Let \( i \in \mathbb{Z}_{\geq 0} \) be a position, \( p \) a proposition, \( s \) a nominal, \( d \) a value in the domain of parameters, and \( \varphi_1 \) and \( \varphi_2 \) any HL formulae. The satisfaction relation \( (\tau, i, \mathcal{E}) \models \varphi \) is defined inductively as follows:

\[
\begin{align*}
(\tau, i, \mathcal{E}) & \not\models \bot; \\
(\tau, i, \mathcal{E}) & \models p \text{ iff } p \in e[i]; \\
(\tau, i, \mathcal{E}) & \models (\varphi_1 \Rightarrow \varphi_2) \text{ iff } (\tau, i, \mathcal{E}) \models \varphi_1 \text{ implies } (\tau, i, \mathcal{E}) \models \varphi_2; \\
(\tau, i, \mathcal{E}) & \models (\varphi_1 \bigcup \varphi_2) \text{ iff there exists } j > i \text{ with } (\tau, j, \mathcal{E}) \models \varphi_2 \text{ and for all } i < j' < j \text{ it holds that } (\tau, j', \mathcal{E}) \models \varphi_1; \\
(\tau, i, \mathcal{E}) & \models \pi_1 \sim \pi_2 \text{ iff } E(\pi_1) \sim E(\pi_2); \\
(\tau, i, \mathcal{E}) & \models x \downarrow s. \varphi \text{ iff } (\tau, i, \mathcal{E}[x := d]) \models \varphi, \text{ where } (s, d) \in P[i].
\end{align*}
\]

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 \( \text{COND} \) of condition structures and a set \( \text{ASSI} \) 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 \( \text{COND}_{NFV} \) and \( \text{COND}_{FV} \), the following formulae are defined.

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

\[
\varnothing(\text{cond}) \triangleq \Box (x_1 \downarrow s_1 \cdots x_n \downarrow s_n. (e \Rightarrow \text{prop}(x_1, ..., x_n))).
\]

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 \( \text{prop}(d_1, ..., d_n) \) is true with \( \{(s_1, d_1), ..., (s_n, d_n)\} \subseteq \rho[z1] \).

- For a condition structure \( \text{cond} \in \text{COND}_{FV} \), there is a tuple \( \alpha(\text{cond}) \triangleq (\text{prop}(s_{z1}, ..., s_{zm}, v_{z1x1}, ..., v_{zxm}), o, \text{ASSI}(v_{z1x1}, ..., v_{zxm}) \text{ with } \text{ASSI}(v_{z1x1}, ..., v_{zxm}) \triangleq \{\text{ass}(v_{z1x1}), ..., \text{ass}(v_{zxm})\}) \text{ and } \text{ass}(v_{zxi}) \triangleq (v_{zxi}, s_{zxi}, o') \text{ for any } 1 \leq i \leq m. \text{ Let } e = \text{lab}(a) \text{ and } e' = \text{lab}(o') \text{ be the events of the two event occurrences, the condition structure is translated into an HL formula}

\[
\varnothing(\text{cond}) \triangleq \Box ((e' \land \varnothing e) \Rightarrow (x_1 \downarrow v_1 \cdots x_m \downarrow v_m. (e' \land \varnothing y_1 \downarrow s_1 \cdots y_n \downarrow s_n. (e \land (\text{prop}(y_1, ..., y_n, x_1, ..., x_m)))).
\]

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 $\text{prop}(d_{zy_1}, ..., d_{zy_n}, d_{zx_1}, ..., d_{zx_m})$ is true with
\[\{(s_{zy_1}, d_{zy_1}), ..., (s_{zy_n}, d_{zy_n})\} \subseteq \rho[z_2] \text{ and } \{(s_{zx_1}, d_{zx_1}), ..., (s_{zx_m}, d_{zx_m})\} \subseteq \rho[z_1].\]

From a PeLSC $PU \triangleq (Uch, \text{COND}, \text{ASSI})$ we define an HL formula
\[
\varphi(\text{PU}) \triangleq \left( \varphi(Uch) \land \bigwedge_{\text{cond} \in \text{COND}_{NPV}} \vartheta(\text{cond}) \land \bigwedge_{\text{cond} \in \text{COND}_{FV}} \vartheta(\text{cond}) \right).
\]

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

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 $\tau$ satisfies $\varphi$ 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 $\varphi$ 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 $\tau$ is essentially the same with the process of checking an LTL formula over an event trace $\sigma$. Since the complexity of checking whether or not a trace $\sigma$ satisfies an LTL formula is linear with respect to the size of the trace $\sigma$, 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.
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


