Interaction-based Offline Runtime Verification of Distributed Systems

Erwan Mahe[1], Boutheina Bannour[1], Christophe Gaston[1],
Arnault Lapitre[1], and Pascale Le Gall[2]

[1] Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
[2] Université Paris-Saclay, CentraleSupélec, F-91192, Gif-sur-Yvette, France

Abstract. Interactions are formal models describing asynchronous communications within a distributed system. They can be drawn in the fashion of sequence diagrams and associated with an operational semantics in the style of process algebras. In this paper, we propose an algorithm for offline runtime verification against interactions. Our algorithm deals with observability issues e.g. that some subsystems may not be observed or that some events may not be observed when the end of monitoring on different subsystems cannot be synchronized. We prove the algorithm's correctness and assess the performance of an implementation.

Keywords: distributed systems · offline runtime verification · interaction · multitrace semantics · partial observability

1 Introduction

Context. Distributed Systems (DS) have been identified in the recent survey [26] as one of the most challenging application domains for Runtime Verification (RV). An important bottleneck is that the formal references against which system executions are analyzed are specified using formalisms or logics usually equipped with trace semantics. Indeed, because DS are composed of subsystems deployed on different computers and communicating via message passing, their executions are more naturally represented as collections of traces observed at the level of the different subsystems' interfaces rather than as single global traces [7,24]. Those collections can be gathered using a distributed observation architecture involving several local observation devices, each one dedicated to a subsystem, and deployed on the same computer as the subsystem it is dedicated to. An approach to confront such collections of local execution traces to formal references with a trace semantics might consist in identifying the global traces that result from all possible temporal orderings of the events occurring in the local traces. If none of those global traces conforms to the formal reference, then we might conclude that an error is observed [24]. However, the absence of a global clock implies that, in all generality, it is not possible to synchronize the endings of the different local observation processes. Therefore, in the process of reconstructing global traces, some events might be missing in local traces. Such

problems occur whenever, for technical or legal reasons, it is not possible to observe some subsystems or else the observation has been interrupted too early.

Contributions. In this paper, we propose a Runtime Verification approach dedicated to DS with an emphasis on overcoming issues of *partial observability*, whether due to the absence of a global clock, or to the impossibility of observing some subsystem executions. Our approach belongs to the family of offline RV techniques in which traces are logged prior to their analysis. As for formal references, we inherit the framework of *interaction* models from earlier works [20,18]. Interactions describe actor-oriented scenarios and can be represented graphically in the fashion of UML Sequence Diagrams (UML-SD) [25] or Message Sequence Charts (MSC) [14]. We designed in [18] an algorithm to decide whether or not a collection of local traces is accepted by an interaction. However, this algorithm cannot cope with partial observability. The core contribution of this paper is then to define an algorithm to tackle those limitations, i.e. to deal with collections of local traces with missing or incomplete ones. Theorem 1 will enable us to relate collections of local traces reflecting partially observed executions to those of the original reference interaction. The key operator in our algorithm is a removal operator (Definition 5) discarding parts of the interaction relative to unobserved subsystems. We prove the correctness of our algorithm and argue how the use of the removal operations allows us to solve partial observability (Theorem 2). All proofs are available in [19]. Finally, we present some experiments using an implementation of our algorithm, given as an extension of the HIBOU tool [17].

Related work. Solutions to the oracle problem (offline RV) for DS using local logs often rely on a preliminary reordering of events using either timestamps [24] or some happened-before relations (of Lamport [15]) [16,23,7]. In [11,9,12] such solutions rely on a set of discrete and local behavioral models. DS behaviors are modeled by Input/Output Transition Systems (IOTS) [11,12] or by Communicating Sequential Processes (CSP) [9] and local observations are intertwined to associate them with global traces that can be analyzed w.r.t. models. Those approaches however require to synchronize local observations, based on the states in which each of the logging processes terminates (e.g., based on quiescence states in [11], termination/deadlocks in [9] or pre-specified synchronization points in [12]). The works [10,24,8,13] focus on verifying distributed executions against models of interaction (while [10,13] concern MSC, [24] considers choreographic languages, [8] session types, and [4] trace expressions). [10,24] propose offline RV that relies on synchronization hypotheses and on reconstructing a global trace by ordering events occurring at the distributed interfaces (by exploiting the observational power of testers [10] or timestamp information assuming clock synchronization [24]). Our RV approach for multitraces does not require synchronization prerequisites on DS logging. Thus, unlike previous works on offline RV, we can analyze DS executions without needing a synchronization hypothesis on the ending of local observations. For online RV, the work [13] depends on a global component (network sniffer) while the work [8] proposes local RV against projections of interactions satisfying conditions that enforce intended global behaviors. In contrast to these works, we process collections of

local logs against interactions. The work [4] focuses on how distributed monitors can be adapted for partial observation. Yet, our notion of partial observation is distinct from that of [4] where messages are exchanged via channels which are associated to an observability likelihood. [4] proposes specification transformations by removing or making optional several identified unobservable events. We instead deal with partial observability from the perspective of analyzing truncated multitraces due to synchronization issues.

Paper outline. Section 2 discusses the nature of DS, their modelling with interactions and the challenge of applying RV to DS. Section 3 defines multitraces, interactions and associated removal operations. Section 4 defines and proves the correctness of our RV algorithm. Section 5 reports experimental results.

2   Preliminaries

Notations Given a set A, A^* is the set of words on A, with ε the empty word and the "." concatenation law. For any word $w \in A^*$, $|w|$ is the length of w and any word w' is a prefix of w if there exists a word w'', possibly empty, such that $w = w'.w''$. Let us note $\overline{w}$ the set of prefixes of a word $w \in A^*$ and $\overline{W}$ the set of prefixes of all words of a set $W \subseteq A^*$. Given a set A, $|A|$ designates its cardinal and $\mathcal{P}(A)$ is the set of all subsets of A.

Distributed Systems From a black box perspective, the atomic concept to describe the executions of DS is that of communication *actions* occurring on a subsystem's interface. Here a subsystem refers to a software system deployed on a single machine. Anticipating the use of interactions as models in Section 3.2, a subsystem interface is called a *lifeline* and corresponds to an interaction point on which the subsystem can receive or send some messages. Lifelines are elements of a set $\mathcal{L}$ denoting the universe of lifelines. An action occurring on a lifeline is defined by its kind (emission or reception, identified resp. by the symbols ! and ?) and by the message which it carries. We introduce the universe $\mathcal{M}$ of messages. Executions observed on a lifeline l can be modelled as execution *traces* i.e. sequences of actions. For $l \in \mathcal{L}$, the set $\mathbb{A}_l$ of *actions over l* is $\{l\Delta m \mid \Delta \in \{!, ?\},\ m \in \mathcal{M}\}$ and the set $\mathbb{T}_l$ of *traces over l* is $\mathbb{A}_l^*$. For any $a \in \mathbb{A}_l$ of the form $l?m$ or $l!m$, $\theta(a)$ refers to l.

Fig.1 sketches out an example of DS composed of three remote subsystems, assimilated to their interface `bro`, `pub` and `sub`. This DS implements a simplified publish/subscribe scheme of communications (an alternative to client-server architecture). The publisher `pub` may publish messages on the broker `bro` which may then forward them to the subscriber `sub` if it is already subscribed. Fig.1c depicts an interaction defined between the three lifelines. Each lifeline is depicted by a vertical line labelled by its name at the top. By default, the top to bottom direction represents time passing. A communication action depicted above another one on the same lifeline occurs beforehand. Communication actions are represented by horizontal arrows labelled with the action's message. Whenever an arrow exits (resp. enters) a lifeline, there is a corresponding emission (resp. reception) action at that point on the line. For example,
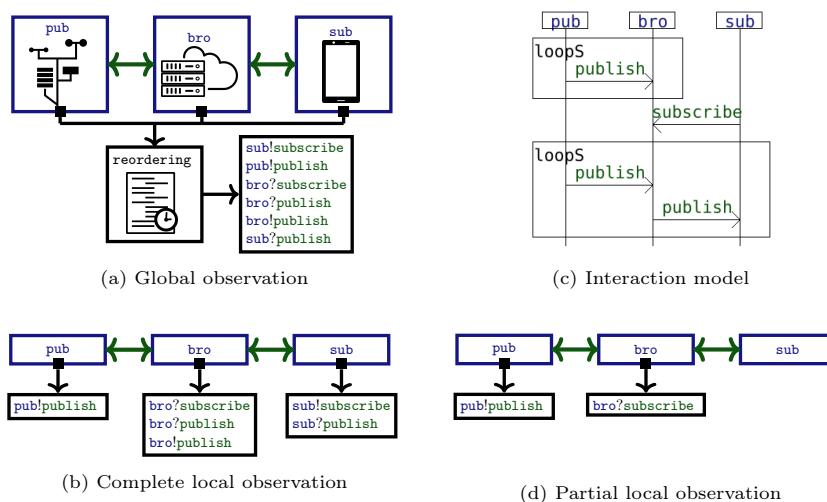
(a) Global observation

(c) Interaction model

(b) Complete local observation

(d) Partial local observation

Fig. 1: A simple publish/subscribe example: architectures & interaction model

the horizontal arrow from the lifeline sub to the lifeline bro indicates that the subsystem sub sends the message subscribe, denoted as sub!subscribe, which is then received by the lifeline bro, denoted as bro?subscribe. More complex behaviors can be introduced through operators (similar to combined fragments in UML-SD) drawn in the shape of boxes that frame sub-behaviors of interest. For instance, in Fig.1c, $loop_S$ corresponds to a sequential loop. From the perspective of the bro lifeline, this implies that it can observe words of the form (bro?publish)*bro?subscribe(bro?publish.bro!publish)* i.e. it can receive an arbitrary number of instances of the publish message then one instance of subscribe and then it can receive and transmit an arbitrary number of publish. A representative global trace specified by the interaction in Fig.1c is (see Fig.1a):

  sub!subscribe.pub!publish.bro?subscribe.bro?publish.bro!publish.sub?publish

This trace illustrates that the pub and sub lifelines can send their respective messages publish and subscribe in any order since there are no constraints on their ordering. In contrast, the reception of a message necessarily takes place after its emission. Since the reception of the message subscribe takes place before that of the publish message, this last message necessarily corresponds to the one occurring in the bottom loop. The global trace in Fig.1a is a typical example of a trace *accepted* by the interaction in Fig.1c. Indeed, this trace realizes one of the behaviors specified by the interaction which corresponds to: unfolding zero times the first loop; realizing the passing of the message subscribe between lifelines sub and bro; unfolding one time the second loop. None of the prefixes of this accepted trace is an accepted trace.

Accepted multitraces Following the terminology of [7,18], we call *multitrace* a collection of local traces, one per remote subsystem. Fig.1b depicts a multitrace involving 3 local traces: bro?subscribe.bro?publish.bro!publish for subsys-

tem `bro`, `pub!publish` for `pub`, and `sub!subscribe.sub?publish` for `sub`. It is possible to interleave these local traces to obtain the global trace in Fig.1a, i.e. the multitrace in Fig.1b corresponds to the tuple of projections of the global trace in Fig.1a onto each of the sub-systems. The tuple of projections of a global trace is unique. However, conversely, one might compute several global traces associated to the same tuple of local traces. This is because, in all generality, there is no ordering between actions occurring on different lifelines. For example, from the multitrace of Fig.1b, one could reconstruct the global trace:

`pub!publish.sub!subscribe.bro?subscribe.bro?publish.bro!publish.sub?publish`

The tuple of projections of this global trace is also the multitrace in Fig.1b. With the algorithm from [18] one can recognize exactly accepted multitraces (e.g. the one from Fig.1b), which correspond to projections of accepted global traces (e.g. Fig.1a).

Logging and Partial observability Offline RV requires to collect execution traces prior to their analyses. In this process, it might be so that some subsystems cannot be equipped with observation devices. Moreover, due to the absence of synchronization between the local observations, the different logging processes might cease at uncorrelated moments. For example, let us consider the multitrace in Fig.1d as an observed execution of the system considered in Fig.1, where, by hypothesis, the subsystem `sub` is not observed. Remark that this multitrace corresponds to a partial observation of the multitrace in Fig.1b. Indeed, each trace corresponding to a given subsystem in Fig.1d is a prefix of the trace corresponding to the same sub-system in Fig.1b. Thus, if `sub` executions were also observed and with longer observation times for each local observation processes, it may well be that one would have observed the multitrace in Fig.1b rather than the one in Fig.1d. when analyzing the multitrace in Fig.1d against the interaction in Fig.1c, we need the RV process not to conclude on the occurrence of an error. Hence, we want to be able to recognize multitraces in which each of the local traces can be extended to reconstruct a multitrace accepted by the interaction. Concretely, this means recognizing *multi-prefixes* of accepted multitraces. Let us remark that a projection of a prefix of an accepted global trace is a multi-prefix of an accepted multitrace. However the reverse is not true. For example, there exists no prefix of a global trace accepted by the interaction in Fig.1c that projects on the multitrace in Fig.1d. This is because the emission of `subscribe` by `sub` would precede its reception by `bro` in any accepted global trace. However, this emission is not observed in the multitrace in Fig.1d. Therefore, dealing with partial observability does not boil down to a simple adaptation of the algorithm in [18]. In this paper, the aforementioned two types of partial observation (unobserved subsystems and early interruption of observation) will be approached in the same manner, noting in particular that an empty local trace can be seen both as missing and incomplete. The key mathematical operator used for that purpose consists in the removal of a lifeline from both interactions and multitraces. This operator allows us to define an algorithm for recognizing multi-prefixes of accepted multitraces while avoiding the complex search for a matching global execution, taking into account potential missing actions.

3   Multitraces, interactions, and removal operations

3.1   Multitraces

As outlined in Section 2, a DS is a collection of communicating subsystems, each having a lifeline as local interface. A DS is characterized by a finite set of lifelines $L \subseteq \mathcal{L}$, called a *signature*. For $L \subseteq \mathcal{L}$, $\mathbb{A}(L)$ denotes the set $\cup_{l \in L} \mathbb{A}_l$. Executions of a DS are associated to *multitraces*, i.e. collections of traces, one per lifeline:

Definition 1. *Given $L \subseteq \mathcal{L}$, the set $\mathbb{M}(L)$ of* multitraces *over L is*[3] $\prod_{l \in L} \mathbb{T}_l$. *For $\mu = (t_l)_{l \in L}$ in $\mathbb{M}(L)$, we denote by $\mu_{|l}$ the trace component $t_l \in \mathbb{T}_l$ and by $\overline{\mu} = \{\mu' \mid \mu' \in \mathbb{M}(L), \forall l \in L, \mu'_{|l} \in \overline{\mu_{|l}}\}$ the set of its* multi-prefixes.

Multi-prefixes are extended to sets: $\overline{M}$ is the set of all multi-prefixes of all multitraces in $M \subseteq \mathbb{M}(L)$. We denote by ε_L the empty multitrace in $\mathbb{M}(L)$ defined by $\forall l \in L, \varepsilon_{L|l} = \varepsilon$. Additionally, for any $\mu \in \mathbb{M}(L)$, we use the notations $\mu[t]_l$ to designate the multitrace μ in which the component on l has been replaced by $t \in \mathbb{T}_l$ and $|\mu|$ to designate the cumulative length $|\mu| = \sum_{l \in L} |\mu_{|l}|$ of μ.

As discussed in Section 2, two communication actions occurring on different traces of a multitrace cannot be temporally ordered. Likewise, when several subsystems are observed concurrently, there is no way to synchronize the endings of their observations. So, any multitrace $\mu' \in \overline{\mu}$ can be understood as a partial observation of the execution characterized by μ. An edge case of this partial observation occurs when some of the subsystems are not observed at all, i.e. when some lifelines are missing. The rmv_h function of Definition 2 simply removes the trace concerning the lifeline h from a multitrace.

Definition 2. *For $L \subseteq \mathcal{L}$, the function $\mathsf{rmv}_h : \mathbb{M}(L) \to \mathbb{M}(L \setminus \{h\})$ is s.t.:* $\forall \mu \in \mathbb{M}(L), \mathsf{rmv}_h(\mu) = (\mu_{|l})_{l \in L \setminus \{h\}}$

The function rmv_h is canonically extended to sets. We introduce operations to add an action to the left (resp. right) of a multitrace. For the sake of simplicity, we use the same symbol ^ for these left- and right-concatenation operations:

$$\forall a \in \mathbb{A}(L), \forall \mu \in \mathbb{M}(L), \; a\hat{}\mu = \mu[a.\mu_{|\theta(a)}]_{\theta(a)} \quad \text{and} \quad \mu\hat{}a = \mu[\mu_{|\theta(a)}.a]_{\theta(a)}$$

Note that for any μ and a, we have $|\mu\hat{}a| = |a\hat{}\mu| = |\mu| + 1$. We extend ^ to sets of multitraces as follows: $a\hat{}T = \{a\hat{}\mu \mid \mu \in T\}$ and $T\hat{}a = \{\mu\hat{}a \mid \mu \in T\}$.

For two multitraces μ_1 and μ_2 in $\mathbb{M}(L)$:

- $\mu_1 \cup \mu_2$ denotes the alternative defined as follows: $\mu_1 \cup \mu_2 = \{\mu_1, \mu_2\}$;
- $\mu_1 ; \mu_2$ denotes their sequencing defined as follows: if $\mu_2 = \varepsilon_L$ then $\mu_1 ; \mu_2 = \mu_1$ else, μ_2 can be written as $a\hat{}\mu'_2$ and $\mu_1 ; \mu_2 = (\mu_1\hat{}a) ; \mu'_2$;
- $\mu_1 || \mu_2$ denotes their interleaving and is defined as the set of multitraces describing parallel compositions of μ_1 and μ_2:

$$\varepsilon_L || \mu_2 = \{\mu_2\} \qquad\qquad \mu_1 || \varepsilon_L = \{\mu_1\}$$
$$(a_1\hat{}\mu_1) \;||\; (a_2\hat{}\mu_2) \;=\; (a_1\hat{}(\mu_1 \;||\; (a_2\hat{}\mu_2))) \cup (a_2\hat{}((a_1\hat{}\mu_1) \;||\; \mu_2)))$$

---

[3] Given a family $(A_i)_{i \in I}$ of sets indexed by a finite set I, $\prod_{i \in I} A_i$ is the set of tuples $(a_1, \ldots, a_i, \ldots)$ with $\forall i \in I, a_i \in A_i$.

Let us remark that μ' is a prefix of a multitrace μ (i.e. $\mu' \in \overline{\mu}$) iff there exists μ'' verifying $\mu'; \mu'' = \mu$. Operations $\cup$, $;$ and $||$ are extended to sets of multitraces as $\diamond : \mathcal{P}(\mathbb{M}(L))^2 \to \mathcal{P}(\mathbb{M}(L))$ for $\diamond \in \{\cup, ;, ||\}$. Operators $;$ and $||$ being associative, this allows for the definition of repetition operators in the same manner as the Kleene star is defined over the classical concatenation. Given $\diamond \in \{;, ||\}$, the Kleene closure $\diamond^*$ is s.t. for any set of multitraces $T \subseteq \mathbb{M}(L)$ we have:

$$T^{\diamond *} = \bigcup_{j \in \mathbb{N}} T^{\diamond j} \text{ with } T^{\diamond 0} = \{\varepsilon_L\} \text{ and } T^{\diamond j} = T \diamond T^{\diamond(j-1)} \text{ for } j > 0$$

$\mathbb{M}(L)$ fitted with the set of algebraic operators $\mathcal{F} = \{\cup, ;, ||, \ ;^*, \ ||^*\}$ is an $\mathcal{F}$-algebra. The operation rmv_h preserves the algebraic structures between the $\mathcal{F}$-algebras of signatures L and $L \setminus \{h\}$.

Property 1 (Elimination preserves operators). For any μ_1 and μ_2 in $\mathbb{M}(L)$, for any $\diamond \in \{\cup, ;, ||\}$, $\mathsf{rmv}_h(\mu_1 \diamond \mu_2) = \mathsf{rmv}_h(\mu_1) \diamond \mathsf{rmv}_h(\mu_2)$.

Property 1 is obtained directly for the union and by induction for the other cases. Those results can be extended to sets of multitraces and imply that repetitions of those scheduling algebraic operators (with their Kleene closures) are also preserved by the elimination operator rmv_h.

3.2 Interactions

Interaction models, such as the one in Fig.1c, can be formalized as terms of an inductive language. [20,18] consider an expressive language with two sequencing operators, weak and strict, for ordering actions globally. Here, as only collections of remote local traces are considered, weak and strict sequencing can no longer be distin-

Fig. 2: Semantics of example from Fig.1c

guished. This explains why we only consider a unique sequencing operator *seq*. Following the syntax from Definition 3, the interaction term of Fig.1c is:
$seq(\ loop_S(seq(\mathtt{pub!publish}, \mathtt{bro?publish})),\ seq(seq(\mathtt{sub!subscribe}, \mathtt{bro?subscribe}),$
$loop_S(seq(seq(\mathtt{pub!publish}, \mathtt{bro?publish}), seq(\mathtt{bro!publish}, \mathtt{sub?publish}))))).$

Definition 3. *Given signature L, the set $\mathbb{I}(L)$ of interactions over L is the set of ground terms built over the following symbols provided with arities in $\mathbb{N}$:*
– the empty interaction $\varnothing$ and any action a in $\mathbb{A}(L)$ of arity 0;
– the two loop operators $loop_S$ and $loop_P$ of arity 1;
– and the three operators seq, par and alt of arity 2.

The semantics of interactions can be defined as a set of multitraces in a denotational style by associating each syntactic operator with an algebraic counterpart. This is sketched out in Fig.2 in which the semantics of the interaction

given in Fig.1c is given. The denotational formulation, which is compositional, is defined in Definition 4 and illustrated in Fig.1c.

Definition 4 (M-semantics). *Given* $L \subseteq \mathcal{L}$, *the multitrace semantics* $\sigma_{|L}$: $\mathbb{I}(L) \to \mathcal{P}(\mathbb{M}(L))$ *is defined inductively using the following interpretations:*
$-$ $\{\varepsilon_L\}$ *for* $\varnothing$ *and* $\{a\hat{\ }\varepsilon_L\}$ *for* a *in* $\mathbb{A}_L$;
$-$;* *(resp.* $||^*$*) for loop operator* $loop_S$ *(resp.* $loop_P$*);*
$-$; *(resp.* $||$ *and* $\cup$*) for binary operator seq (resp. par and alt).*

Interactions can also be associated with operational semantics in the style of Plotkin [21]. Its definition relies on two predicates denoted by $\downarrow$ and $\to$: for an interaction i, $i \downarrow$ states that $\varepsilon_L \in \sigma_{|L}(i)$ and $i \xrightarrow{a} i'$ states that all multitraces of the form $a\hat{\ }\mu'$ with $\mu' \in \sigma_{|L}(i')$ are multitraces of $\sigma_{|L}(i)$. This operational semantics is equivalent to the denotational formulation.

Property 2 (Operational semantics). There exist a predicate $\downarrow \subseteq \mathbb{I}(L)$ and a relation $\to \subseteq \mathbb{I}(L) \times \mathbb{A}(L) \times \mathbb{I}(L)$ such that, for any $i \in \mathbb{I}(L)$ and $\mu \in \mathbb{M}(L)$, the statement $\mu \in \sigma_{|L}(i)$ holds iff it can be proven using the following two rules:

$$\frac{i \downarrow}{\varepsilon_L \in \sigma_{|L}(i)} \qquad \frac{\mu \in \sigma_{|L}(i') \qquad i \xrightarrow{a} i'}{a\hat{\ }\mu \in \sigma_{|L}(i)}$$

The proof is a transposition for multitrace semantics of the proof in [21] given for global traces. The algebraic characterisation of Definition 4 underpins results involving the use of the rmv_h function while the operational characterisation of Property 2 is required in the definition and proof of the RV algorithm. In this paper, we do not need the inductive definitions

Fig. 3: Removing lifeline `sub`

of $\downarrow$ and $\to$. It suffices to consider their existence (Property 2). In addition, we will use the notation $i \xrightarrow{a}$ (resp. $i \xrightarrow{a}\!\!\!\!\!/\,$) when there exists (resp. does not exist) an interaction i' s.t. $i \xrightarrow{a} i'$.

The removal of lifelines for multitraces (cf. Definition 2) has a counterpart for interactions. On the left of Fig.3 we draw our previous example while highlighting lifeline `sub` which we remove to obtain the interaction on the right. Whenever we remove a lifeline l, the resulting interaction does not contain any action occurring on l. Removal, as defined in[4] Definition 5 in a functional style, preserves the term structure, replacing actions on the removed lifeline with the empty interaction.

Definition 5. *For a signature* $L \subseteq \mathcal{L}$ *and a lifeline* $h \in L$ *we define* rmv_h : $\mathbb{I}(L) \to \mathbb{I}(L \setminus \{h\})$ *s.t. for any interaction* $i \in \mathbb{I}(L)$:

---

[4] We overload the notation rmv_h which applies to both multitraces and interactions.
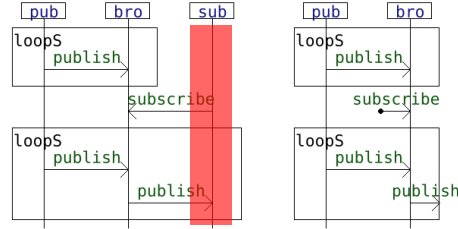
$\mathsf{rmv}_h(i) = $ **match** i **with**
| $\varnothing$ $\to \varnothing$
| $a \in \mathbb{A}(L) \to$ **if** $\theta(a) = h$ **then** $\varnothing$ **else** a
| $f(i_1, i_2)$ $\to f(\mathsf{rmv}_h(i_1), \mathsf{rmv}_h(i_2))$ **for** $f \in \{seq, alt, par\}$
| $loop_k(i_1) \to loop_k(\mathsf{rmv}_h(i_1))$ **for** $k \in \{S, P\}$

Theorem 1 relates the removal operations on multitraces and interactions with one another. The semantics of an interaction i in which we remove lifeline h can be obtained by removing lifeline h from all the multitraces of the semantics of i. This result is obtained reasoning by induction on interaction terms.

**Theorem 1.** *For any signature L, any $i \in \mathbb{I}(L)$ and any $h \in L$:*

$$\sigma_{|L \setminus \{h\}}(\mathsf{rmv}_h(i)) = \mathsf{rmv}_h(\sigma_{|L}(i))$$

4   Offline RV for multitraces

We aim to define a process to analyze a multitrace μ against a reference interaction i, both defined on a common signature L. To check whether or not a multitrace μ is accepted by i, i.e. $\mu \in \sigma_{|L}(i)$, the key principle given in [18] was to find a globally ordered behavior specified by i (via the $\to$ execution relation) that matches μ, i.e. an accepted global trace that can be projected into μ. To do so, it relies on a general rule $(i, a \char`^ \mu') \rightsquigarrow (i', \mu')$ s.t. $i \xrightarrow{a} i'$, i.e. it explores all the actions a directly executable from i and that match the head of a local trace. The analysis is then pursued recursively from (i', μ'), i.e. the multitrace where a has been removed and the follow-up interaction i', until the multitrace is emptied of actions.

For illustrative purposes, let us consider Fig.4 where each square annotated with a circled number corresponds to such a tuple (i, μ), with interaction i drawn on the left and multitrace μ represented on the right. Starting from the tuple indexed by ③, with interaction i_3 and multitrace $\mu_3 = (\varepsilon, \texttt{bro?subscribe})$, one can see that we can reach ④ by both consuming $\texttt{bro?subscribe}$ from μ_3 and executing it in i_3, leading to the tuple (i_4, μ_4) in ④. This transition $(i_3, \texttt{bro?subscribe} \char`^ (\varepsilon, \varepsilon)) \rightsquigarrow (i_4, (\varepsilon, \varepsilon))$ is based on having $i_3 \xrightarrow{\texttt{bro?subscribe}} i_4$. Thus, Fig.4 sketches the construction of a graph whose nodes are pairs of interactions and multitraces and whose arcs are built using the $\rightsquigarrow$ relation.

While in [18], we were interested in solving the membership problem "$\mu \in \sigma_{|L}(i)$", we are now interested in defining an offline RV algorithm. In line with the discussion of Section 2 about partial observability, μ reveals an error if μ is neither in $\sigma_{|L}(i)$ nor can be extended into an element of $\sigma_{|L}(i)$ i.e. μ diverges from i iff $\mu \notin \overline{\sigma_{|L}(i)}$. We introduce a rule involving the removal operation to accommodate the need to identify multi-prefixes of multitraces. Indeed, as the execution relation $\to$ only allows executing actions in the global order in which they are intended to occur, we may reach cases in which the next action which may be consumed in the multitrace cannot be executed due to having a preceding action missing in the multitrace.
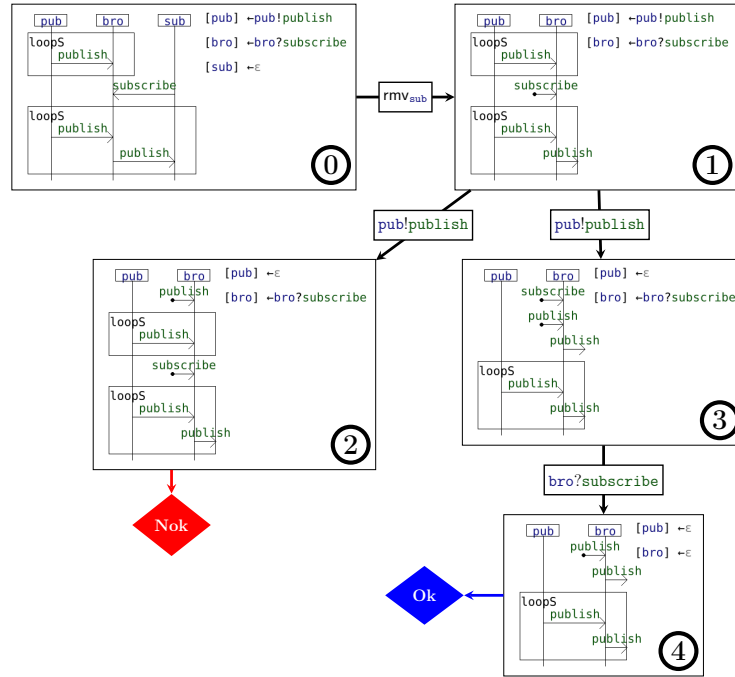
Fig. 4: An exploration s.t. $\omega_L(i,\mu) = Pass$

Let us illustrate this with node ⓪ of Fig.4. bro?subscribe is the first action that occurs on lifeline bro in the multitrace. However, it cannot be executed because it must be preceded by sub!subscribe. Yet, either because the behavior on lifeline sub is not observed or because the logging process ceased too early on sub, it might well be that sub!subscribe occurred in the actual execution although it was not logged. With our new algorithm, because the condition that $\mu_{|\mathsf{sub}} = \varepsilon$ is satisfied, from node ⓪, we apply a rule yielding the transformation $(i,\mu) \rightsquigarrow (\mathsf{rmv}_{\mathsf{sub}}(i), \mathsf{rmv}_{\mathsf{sub}}(\mu))$, removing lifeline sub, which allows us to pursue the analysis from node ①. To summarize, Fig.4 illustrates (part of) the graph that can be constructed from a pair (i_0, μ_0) using the relation $\rightsquigarrow$. We have 5 nodes numbered from 0 (the initial node of the analysis) to 4. Arcs correspond to the consumption of an action, the application of the rmv operator, or the emission of a verdict. The empty multitrace in node ④ allows us to conclude $\mu_0 \in \overline{\sigma_{|L}(i_0)}$.

4.1 The algorithm

As the rmv operator has the effect of changing the signature, we introduce the set $\mathbb{I}_{\mathcal{L}}$ (resp. $\mathbb{M}_{\mathcal{L}}$) to denote the set of all interactions (resp. multitraces) defined on a signature of $\mathcal{L}$. Let us define a directed search graph with vertices either of the form $(i,\mu) \in \mathbb{I}_{\mathcal{L}} \times \mathbb{M}_{\mathcal{L}}$ or one of two specific verdicts Ok and Nok.

We denote by $\mathbb{V}$ the set of all vertices:

$$\mathbb{V} = \{Ok, Nok\} \cup \left(\bigcup_{L \subseteq \mathcal{L}} \mathbb{I}(L) \times \mathbb{M}(L) \right)$$

The arcs of $\mathbb{G}$ are defined by 4 rules: R_o, R_n leading to respectively the sink vertices Ok and Nok, R_e (for "execute") for consuming actions of the multitrace according to the $\rightarrow$ predicate of the operational formulation (cf. Property 2), and R_r (for "removal"), for removing a lifeline from the interaction and multitrace.

Definition 6 (Search graph). $\mathbb{G} = (\mathbb{V}, \rightsquigarrow)$ *is the graph s.t. for all v, v' in $\mathbb{V}$, $v \rightsquigarrow v'$ iff there exists a rule R_x with $x \in \{o, n, e, r\}$ s.t. $(R_x)\frac{v}{v'}$ where rules R_x are defined as follows, with $L \subseteq \mathcal{L}$, $h \in L$, $i, i' \in \mathbb{I}(L)$, and $\mu, \mu' \in \mathbb{M}(L)$:*

$$(R_o) \frac{i \quad \varepsilon_L}{Ok} \qquad\qquad (R_r) \frac{i \quad \mu}{\mathsf{rmv}_h(i) \quad \mathsf{rmv}_h(\mu)} \left\{ \begin{array}{l} (\mu \neq \varepsilon_L) \wedge \\ (\mu_{|h} = \varepsilon) \end{array} \right.$$

$$(R_e) \frac{i \quad \mu}{i' \quad \mu'} \left\{ \begin{array}{l} \exists a \in \mathbb{A}(L), \\ \mu = a \hat{} \mu' \wedge i \xrightarrow{a} i' \end{array} \right. \qquad (R_n) \frac{i \quad \mu}{Nok} \left\{ \begin{array}{l} (\forall l \in L, \mu_{|l} \neq \varepsilon) \wedge \\ \left(\begin{array}{l} \forall a \in \mathbb{A}(L), \forall \mu' \in \mathbb{M}(L), \\ \mu = a \hat{} \mu' \Rightarrow i \not\xrightarrow{a} \end{array} \right) \end{array} \right.$$

Rules R_e and R_r specify edges of the form $(i, \mu) \rightsquigarrow (i', \mu')$ with i' and μ' defined on the same signature: the application of R_e corresponds to the simultaneous consumption of an action at the head of a component of μ and the execution of a matching action in i while the application of R_r corresponds to the removal of a lifeline h s.t. $\mu_{|h} = \varepsilon$. Moreover, vertices of the form (i, μ) are not sinks of $\mathbb{G}$. Indeed, if $\mu = \varepsilon_L$ then R_o can apply, otherwise $\mu \neq \varepsilon_L$ and: **(1)** if at least a component $\mu_{|h}$ of μ is empty, then rule R_r can apply. **(2)** if there is a match between an action that can be executed from i and the head of a component of the multitrace then rule R_e can apply. **(3)** if both conditions 1 and 2 do not hold then rule R_n applies.

Proving $\mu \in \overline{\sigma_{|L}(i)}$, amounts to exhibiting a path in $\mathbb{G}$ starting from (i, μ) and leading to the verdict Ok. Fig.4 depicts such a path for the multitrace $\mu_0 = (\texttt{pub!publish}, \texttt{bro?subscribe}, \varepsilon)$ w.r.t. the interaction i_0 of node ⓪. A first step (application of R_r) removes lifeline $\texttt{sub}$ leading to node ①. This is possible because $\mu_{|\texttt{sub}} = \varepsilon$. From there, by applying rule R_e, the execution of $\texttt{pub?publish}$ allows to reach either node ② or node ③ depending on the loop used. From node ③, the previous removal of lifeline $\texttt{sub}$ has unlocked the execution of $\texttt{bro?subscribe}$ (application of R_e). What remains is ε_L, and hence we can apply rule R_o. From the existence of this path leading to Ok we conclude that μ_0 is a multi-prefix of a multitrace of the interaction depicted in Fig.1c.

Property 3 (Finite search space). Let $L \subseteq \mathcal{L}$, $\mu \in \mathbb{M}(L)$ and $i \in \mathbb{I}(L)$. The sub-graph of $\mathbb{G}$ of all vertices reachable from (i, μ) is finite.

We establish this property by using a measure $|v|$ defined on the vertices v in $\mathbb{V}$ by $|v| = 0$ if $v \in \{Ok, Nok\}$ and $|v| = |\mu| + |L| + 1$ if $s = (i, \mu) \in \mathbb{I}(L) \times \mathbb{M}(L)$.

Definition 7 (Multitrace analysis). *For any $L \subset \mathcal{L}$, we define $\omega_L : \mathbb{I}(L) \times \mathbb{M}(L) \to \{Pass, Fail\}$ s.t. for any $i \in \mathbb{I}(L)$ and $\mu \in \mathbb{M}(L)$:*
$- \omega_L(i, \mu) = Pass$ iff there exists a path in $\mathbb{G}$ from (i, μ) to Ok
$- \omega_L(i, \mu) = Fail$ otherwise

Given Property 3, Definition 7 is well founded insofar as the sub-graph of $\mathbb{G}$ issued from any pair (i, μ) of $\mathbb{V}$ is finite and all paths from (i, μ) can be extended until reaching a verdict (Ok or Nok). Then, we need to prove that the existence of a path from (i, μ) to Ok guarantees that μ is a prefix of a multitrace of i, and that the non-existence of such a path guarantees that μ is not such a prefix. By reasoning by induction on the measure of the vertices of $\mathbb{G}$, we can establish:

Theorem 2. *For any $i \in \mathbb{I}(L)$ and any $\mu \in \mathbb{M}(L)$:*

$$\left(\mu \in \overline{\sigma_{|L}(i)} \right) \Leftrightarrow \left(\omega_L(i, \mu) = Pass \right)$$

4.2 Considerations on implementation

Using a reduction of the 3 SAT problem inspired by [3,18], we can state that the problem of recognizing correct multi-prefixes w.r.t. interactions is NP-hard:
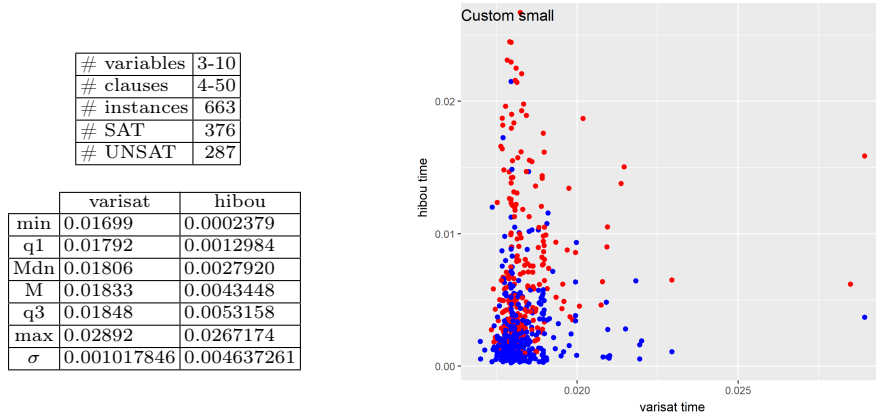
Property 4. The problem of determining whether or not $\mu \in \overline{\sigma_{|L}(i)}$ is NP-hard.

Given the NP-hardness of the underlying problem, the implementation of our algorithm, which relies on the exploration of a graph $\mathbb{G}$, uses additional techniques to reduce the average complexity. Such techniques may include means to cut parts of the graph, the use of pertinent search strategies and priorities for the application of the rules. For instance, if R_r is applicable from a node (i, μ), we can apply rmv on all lifelines which can be removed simultaneously. Also, if both R_r and R_e are applicable from that same node, we can choose not to apply R_e. Those two points are respectively justified by a property of commutativity for rmv (i.e. $\mathsf{rmv}_h \circ \mathsf{rmv}_{h'} = \mathsf{rmv}_{h'} \circ \mathsf{rmv}_h$) and a confluence property for $\rightsquigarrow$ (i.e. if $(i, \mu) \overset{*}{\rightsquigarrow} Ok$ and $(i, \mu) \rightsquigarrow (\mathsf{rmv}_h(i), \mathsf{rmv}_h(\mu))$ then $(\mathsf{rmv}_h(i), \mathsf{rmv}_h(\mu)) \overset{*}{\rightsquigarrow} Ok$).
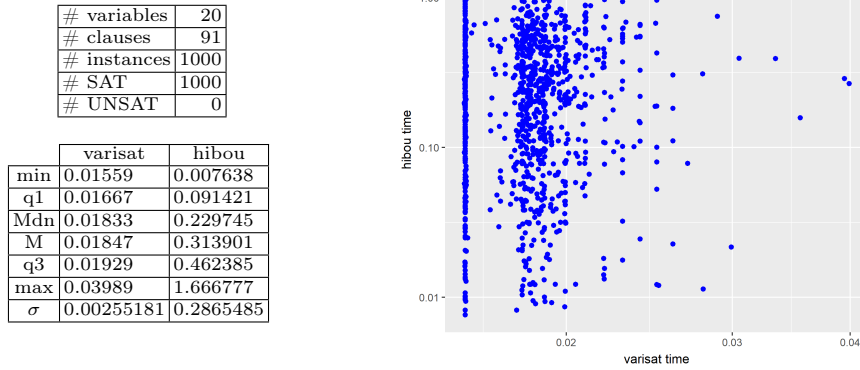
5 Experimental assessment

5.1 3 SAT benchmarks

We have implemented our approach as an extension of the tool HIBOU [17]. In light of Property 4, we have compared the results HIBOU obtained on translated three SAT problems against those of an SAT solver (Varisat [2]). We have used three sets of problems: two custom benchmarks with randomly generated problems and the UF20 benchmark [1]. Fig.5 provides details on 2 benchmarks with, on the top left, information about the input problems (numbers of variables, clauses, instances), on the bottom left statistical information about the time required for the analysis using each tool, and, on the right a corresponding scatter plot. In the plot, each point corresponds to a given 3-SAT problem, with its position corresponding to the time required to solve it. Points in red are unsatisfiable problems while those in blue are satisfiable.

# variables	3-10
# clauses	4-50
# instances	663
# SAT	376
# UNSAT	287

	varisat	hibou
min	0.01699	0.0002379
q1	0.01792	0.0012984
Mdn	0.01806	0.0027920
M	0.01833	0.0043448
q3	0.01848	0.0053158
max	0.02892	0.0267174
σ	0.001017846	0.004637261

(a) Input problems and output results for 'small' custom benchmark

# variables	20
# clauses	91
# instances	1000
# SAT	1000
# UNSAT	0

	varisat	hibou
min	0.01559	0.007638
q1	0.01667	0.091421
Mdn	0.01833	0.229745
M	0.01847	0.313901
q3	0.01929	0.462385
max	0.03989	1.666777
σ	0.00255181	0.2865485

(b) Input problems and output results for UF-20 benchmark

Fig. 5: Experiments on 3SAT benchmarks (times in seconds)

5.2 Use cases experiments

To consider concrete and varied interactions, we experiment with four examples: a protocol for purchasing books [4], a system for querying complex sensor data [5], the Alternating Bit Protocol [22] and a network for uploading data to a server [6]. Fig.6 partially reports on those experiments. For each example, we generated randomly accepted multitraces (ACPT) up to some depth, for which we then randomly selected prefixes (PREF). For each such prefix, we then performed mutations of three kinds: swapping actions (SACT), swapping trace components (SCMP) and inserting noise (NOIS). We report for each category of multitraces times to compute verdicts in Fig.6. As expected, running the algorithm on those multitraces recognizes prefixes and mutants which go out of specification.
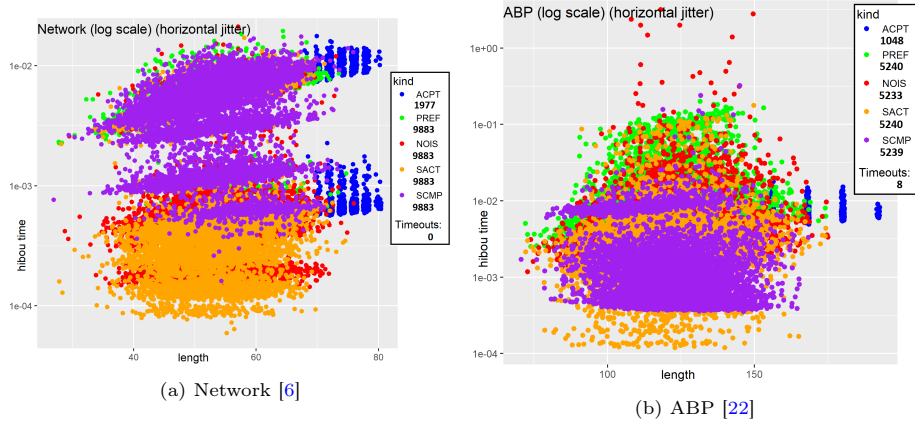
(a) Network [6]

(b) ABP [22]

Fig. 6: Experimental data on a selection of use cases (times in seconds)

6   Conclusion

We have proposed an algorithm for offline RV from multitraces (sets of local execution logs collected on the DS) against interaction models (formal specifications akin to UML-SD/MSC). These multitraces can be partial views of DS executions because some components may either not be observed at all or their observation may have ceased too early. We have proved the correctness of our algorithm which boils down to a graph search. This search is based on two principles, either we match actions of the interaction against those of the input multitrace, or we apply a removal operation on multitraces and interactions. Removal steps allow dealing with observability via disregarding components which are no longer observed parts of the interaction. Future works include other uses of the removal operator (e.g. for performance improvements on RV).

References

1. SATLIB - Benchmark. https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html
2. Varisat CDCL solver. https://docs.rs/varisat/latest/varisat/
3. Alur, R., Etessami, K., Yannakakis, M.: Realizability and verification of MSC graphs. In: Automata, Languages and Programming, 28th Int. Colloquium, ICALP 2001. LNCS, vol. 2076, pp. 797–808. Springer (2001)
4. Ancona, D., Ferrando, A., Franceschini, L., Mascardi, V.: Coping with bad agent interaction protocols when monitoring partially observable multiagent systems. In: Advances in Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection. pp. 59–71. Springer (2018)

5. Bakillah, M., Liang, S., Zipf, A., Mostafavi, M.A.: A dynamic and context-aware semantic mediation service for discovering and fusion of heterogeneous sensor data. Journal of Spatial Information Science **6**, 155–185 (06 2013)
6. Bejleri, A., Domnori, E., Viering, M., Eugster, P., Mezini, M.: Comprehensive multiparty session types. The Art, Science, and Engineering of Programming **3** (02 2019)
7. Benharrat, N., Gaston, C., Hierons, R.M., Lapitre, A., Le Gall, P.: Constraint-based oracles for timed distributed systems. In: Testing Software and Systems. pp. 276–292. Springer (2017)
8. Bocchi, L., Chen, T., Demangeon, R., Honda, K., Yoshida, N.: Monitoring networks through multiparty session types. Theor. Comput. Sci. **669**, 33–58 (2017)
9. Cavalcanti, A., Gaudel, M., Hierons, R.M.: Conformance relations for distributed testing based on CSP. In: IFIP ICTSS. LNCS, vol. 7019, pp. 48–63. Springer (2011)
10. Dan, H., Hierons, R.M.: The oracle problem when testing from MSCs. Comput. J. **57**(7), 987–1001 (2014)
11. Hierons, R.M., Merayo, M.G., Núñez, M.: Controllable test cases for the distributed test architecture. In: ATVA. LNCS, vol. 5311, pp. 201–215. Springer (2008)
12. Hierons, R.M., Merayo, M.G., Núñez, M.: Scenarios-based testing of systems with distributed ports. Softw. Pract. Exp. **41**(10), 999–1026 (2011)
13. Inçki, K., Ari, I.: A novel runtime verification solution for IoT Systems. IEEE Access **6**, 13501–13512 (2018)
14. ITU: Message Sequence Chart (MSC), http//www.itu.int/rec/T-REC-Z.120
15. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. In: Concurrency: the Works of Leslie Lamport, pp. 179–196. ACM (2019)
16. Mace, J., Roelke, R., Fonseca, R.: Pivot tracing: dynamic causal monitoring for distributed systems. In: SOSP. pp. 378–393. ACM (2015)
17. Mahé, E.: Hibou tool. github.com/erwanM974/hibou_label (2022)
18. Mahé, E., Bannour, B., Gaston, C., Lapitre, A., Le Gall, P.: A small-step approach to multi-trace checking against interactions. p. 1815–1822. SAC '21, ACM (2021)
19. Mahé, E., Bannour, B., Gaston, C., Lapitre, A., Le Gall, P.: Dealing with observability in interaction-based offline runtime verification of distributed systems. CoRR (2022), https://arxiv.org/abs/2212.09324
20. Mahé, E., Gaston, C., Le Gall, P.: Revisiting semantics of interactions for trace validity analysis. In: FASE. LNCS, vol. 12076, pp. 482–501. Springer (2020)
21. Mahé, E., Gaston, C., Le Gall, P.: Equivalence of denotational and operational semantics for interaction languages. In: TASE. pp. 113–130. Springer (2022)
22. Mauw, S., Reniers, M.A.: High-level message sequence charts. In: SDL Forum. pp. 291–306. Elsevier (1997)
23. Neves, F., Machado, N., Pereira, J.: Falcon: A practical log-based analysis tool for distributed systems. In: DSN. pp. 534–541. IEEE Computer Society (2018)
24. Nguyen, H.N., Poizat, P., Zaïdi, F.: Passive conformance testing of service choreographies. In: ACM SAC 2012. pp. 1528–1535 (2012)
25. OMG: Unified Modeling Language, http://www.uml.org
26. Sánchez, C., Schneider, G., Ahrendt, W., Bartocci, E., Bianculli, D., Colombo, C., Falcone, Y., Francalanza, A., Krstic, S., Lourenço, J.M., Nickovic, D., Pace, G.J., Rufino, J., Signoles, J., Traytel, D., Weiss, A.: A survey of challenges for runtime verification from advanced application domains (beyond software). Formal Methods Syst. Des. **54**(3), 279–335 (2019)