Offline Trace Checking of Quantitative Properties of Service-Based Applications

Domenico Bianculli
SnT Centre - University of Luxembourg, Luxembourg
Email: domenico.bianculli@uni.lu

Carlo Ghezzi Srđan Krstić Pierluigi San Pietro
DEEP-SE group - DEIB - Politecnico di Milano, Italy
Email:{carlo.ghezzi,srdan.krstic,pierluigi.sanpietro}@polimi.it

***Abstract*—Service-based applications are often developed as compositions of partner services. A service integrator needs precise methods to specify the quality attributes expected by each partner service, as well as effective techniques to verify these attributes. In previous work, we identified the most common specification patterns related to provisioning service-based applications and developed an expressive specification language (SOLOIST) that supports them. SOLOIST is an extension of metric temporal logic with aggregate temporal modalities that can be used to write quantitative temporal properties.**

In this paper we address the problem of performing offline checking of service execution traces against quantitative requirements specifications written in SOLOIST. We present a translation of SOLOIST into CLTLB($\mathscr{D}$), a variant of linear temporal logic, and reduce the trace checking of SOLOIST to bounded satisfiability checking of CLTLB($\mathscr{D}$), which is supported by ZOT, an SMT-based verification toolkit. We detail the results of applying the proposed offline trace checking procedure to different types of traces, and compare its performance with previous work.

I. Introduction

Service-based applications (SBAs) are one of the main approaches followed nowadays to develop modern enterprise information systems, adopting the paradigm of service-oriented computing [1]. SBAs are usually defined as service compositions, created by orchestrating several existing services, possibly provided by third-parties, by means of dedicated languages such as BPEL. Developing and operating an SBA involves many stakeholders: service end-users, the developers and providers of services used in the SBA, as well as the service integrators that realize the composite services. However, service integrators have the ultimate responsibility for maintaining an adequate level of quality attributes (e.g., in terms of functional correctness and QoS, quality of service) of the composite services they provide, independently of (but at the same time, based on) the guarantees and the service-level agreements offered by the providers of the services they compose. This can be achieved in a systematic and formal way by developing a specification language that can capture useful properties of SBAs and by providing means for verifying SBAs against properties written in such a specification language.

Several verification techniques have been developed and tailored [2], [3], [4], [5] for the domain of SBAs, to assist service integrators in verification activities both at design time (e.g., testing, model checking) and run time (e.g., monitoring).

This work has been partially supported by the National Research Fund, Luxembourg (FNR/P10/03).

In the case of formal approaches, the verification techniques adopt a temporal logic (such as LTL, CTL) as the specification language of the properties of interest. In the domain of composite SBAs, these properties express constraints on the interactions of the composite service with its partner services. In a previous work some of the authors developed SOLOIST (*SpecificatiOn Language fOr servIce compoSitions inTeractions*) [6] a metric temporal logic with new, additional temporal modalities that can express properties of SBAs in terms of bounds on some aggregated values, calculated over a certain time window. These modalities have been defined based on an extensive field study [7] of the requirements specifications in the context of service-based applications, and they are tailored to express the most common requirements occurring in practice. The study — performed in collaboration with an industrial partner — analyzed more than 900 requirements specifications, extracted both from research papers and industrial data, and led to the identification of a new class of specification patterns, specific to the domain of service provisioning. Examples of these patterns are those characterizing the *average response time* of a service invocation and the *count/average/maximum number of event occurrences* in a given time window.

In this paper we focus on the problem of performing *offline checking* of execution traces against requirements specifications written in SOLOIST. Trace checking (also called *trace validation* [8] or *history checking* [9]) is a procedure for evaluating a formal specification over a log of recorded events produced by a system, i.e., over a temporal evolution of the system. We assume that a trace is finite and composed by the events corresponding to the interactions of a composite service with its external services (e.g., invoking external service operations or receiving service requests). Traces can be produced by a proper monitoring/logging infrastructure, and made available at the end of the execution to perform offline trace checking.

The main contribution of the paper is an offline trace checking procedure for SOLOIST properties exploiting a translation into CLTLB($\mathscr{D}$) [10], an extension of PLTLB (Propositional Linear Temporal Logic with both past and future modalities) augmented with atomic formulae built over a constraint system $\mathscr{D}$. We chose CLTLB($\mathscr{D}$) as the target of our translation since it supports the definition of arithmetical constraints over a set of integer variables (also called counters); as we will detail in Sect. III, these counters allow a compact and easy-to-verify translation. We express the problem of trace checking of SOLOIST properties in terms of bounded satisfiability checking (BSC) of CLTLB($\mathscr{D}$) and rely on the BSC procedure for metric temporal logic [11] implemented in ZOT. We focus

on requirements containing quantitative properties involving aggregate operations on events occurring in a given time window, like the average response time of a certain operation provided by a partner service.

In the original definition of SOLOIST [6] we showed how, under certain assumptions, the language can be translated into LTL, guaranteeing its decidability based on well-known results in temporal logic. However, this translation was only a proof of concept and was not meant to guarantee efficiency if one would use LTL-based verification procedures. In previous work [12] we introduced a trace checking procedure, based on another encoding of SOLOIST properties into formulae of QF-EUFIDL, the theory of quantifier-free integer difference logic with uninterpreted function and predicate symbols. This encoding was tailored for sparse traces, i.e., traces in which the number of time instants when events occur is very low with respect to the length of the trace. In contrast, the new encoding proposed in this paper supports a much more efficient checking of dense traces. In this paper we also compare the two approaches on traces of different degrees of sparseness.

The rest of the paper is organized as follows[1]. Section II provides background information on SOLOIST and CLTLB($\mathscr{D}$). We present the translation of SOLOIST into CLTLB($\mathscr{D}$) in Section III. In section IV we discuss some implementation details. Section V reports on the evaluation performed to assess the scalability of our approach, also in comparison with previous work. Section VI surveys related work, and Sect. VII concludes the paper, giving some directions for future work.

II. Preliminaries

A. SOLOIST in a Nutshell

In this section we provide a brief overview of SOLOIST; a rationale of the language and a detailed explanation of its semantics are in [6].

The syntax of SOLOIST is defined by the following grammar:

$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \mathsf{U}_I \phi \mid \phi \mathsf{S}_I \phi \mid \mathfrak{C}^K_{\bowtie n}(\phi) \mid \mathfrak{U}^{K,h}_{\bowtie n}(\phi) \mid \mathfrak{M}^{K,h}_{\bowtie n}(\phi) \mid \mathfrak{D}^K_{\bowtie n}(\phi,\phi)$

where $p \in \Pi$, with Π being a finite set of atoms; I is a nonempty interval over $\mathbb{N}$; n, K, h range over $\mathbb{N}$; $\bowtie \in \{<, \leq, \geq, >, =\}$. The arguments ϕ of modalities $\mathfrak{C}, \mathfrak{U}, \mathfrak{M}, \mathfrak{D}$ are restricted to atoms in Π. Moreover, the two arguments in the $\mathfrak{D}$ modality are required to be different atoms.

The U_I and S_I modalities are, respectively, the metric "*Until*" and "*Since*" operators. Additional temporal modalities can be derived using the usual conventions; for example "*Always*" is defined as $\mathsf{G}_I\phi \equiv \neg(\top \mathsf{U}_I \neg\phi)$ and "*Eventually in the Past*" as $\mathsf{P}_I\phi \equiv \top \mathsf{S}_I \phi$, where $\top$ means "true". The remaining modalities are called *aggregate* modalities. The $\mathfrak{C}^K_{\bowtie n}(\phi)$ modality states a bound (represented by $\bowtie n$) on the number of occurrences of an event ϕ in the previous K time instants; it is used to count the number of events in a given time window. The $\mathfrak{U}^{K,h}_{\bowtie n}(\phi)$ (respectively, $\mathfrak{M}^{K,h}_{\bowtie n}(\phi)$) modality expresses a bound on the average (respectively, maximum) number of occurrences of an event ϕ, aggregated over the set of right-aligned adjacent non-overlapping subintervals within a time window K; as in "the average/maximum number of events per hour in the last ten hours". A subtle difference in the semantics of the $\mathfrak{U}$ and $\mathfrak{M}$ modalities is that $\mathfrak{M}$ considers events in the (possibly empty) tail interval, i.e., the leftmost observation subinterval whose length is less than h, while the $\mathfrak{U}$ modality ignores them. The $\mathfrak{D}^K_{\bowtie n}(\phi,\psi)$ modality expresses a bound on the average time elapsed between a pair of specific adjacent events ϕ and ψ occurring in the previous K time instants; it is used to express the concept of average response time (of a service), with ϕ and ψ representing, respectively the start and end events, of a (synchronous) service invocation[2].

$$
\begin{array}{lll}
(w,i) \models p & \text{iff} & p \in \sigma_i \\
(w,i) \models \neg\phi & \text{iff} & (w,i) \not\models \phi \\
(w,i) \models \phi \wedge \psi & \text{iff} & (w,i) \models \phi \wedge (w,i) \models \psi \\
(w,i) \models \phi \mathsf{S}_I \psi & \text{iff} & \text{for some } j < i, \tau_i - \tau_j \in I, (w,j) \models \psi \text{ and for all } k, j < k < i, (w,k) \models \phi \\
(w,i) \models \phi \mathsf{U}_I \psi & \text{iff} & \text{for some } j > i, \tau_j - \tau_i \in I, (w,j) \models \psi \text{ and for all } k, i < k < j, (w,k) \models \phi \\
(w,i) \models \mathfrak{C}^K_{\bowtie n}(\phi) & \text{iff} & c(\tau_i - K, \tau_i, \phi) \bowtie n \text{ and } \tau_i \geq K \\
(w,i) \models \mathfrak{U}^{K,h}_{\bowtie n}(\phi) & \text{iff} & \frac{c(\tau_i - \lfloor \frac{K}{h} \rfloor h, \tau_i, \phi)}{\lfloor \frac{K}{h} \rfloor} \bowtie n \text{ and } \tau_i \geq K \\
(w,i) \models \mathfrak{M}^{K,h}_{\bowtie n}(\phi) & \text{iff} & \max\left\{ \bigcup_{m=0}^{\lfloor \frac{K}{h} \rfloor} \{c(lb(m), rb(m), \phi)\} \right\} \bowtie n \text{ and } \tau_i \geq K \\
(w,i) \models \mathfrak{D}^K_{\bowtie n}(\phi,\psi) & \text{iff} & \frac{\sum_{(s,t) \in d(\phi,\psi,\tau_i,K)} (\tau_t - \tau_s)}{|d(\phi,\psi,\tau_i,K)|} \bowtie n \text{ and } \tau_i \geq K \text{ and } d(\phi,\psi,\tau_i,K) \neq \emptyset
\end{array}
$$

where $c(\tau_a, \tau_b, \phi) = |\{s \mid \tau_a < \tau_s \leq \tau_b \text{ and } (w,s) \models \phi\}|$, $lb(m) = \max\{\tau_i - K, \tau_i - (m+1)h\}$, $rb(m) = \tau_i - mh$, and $d(\phi,\psi,\tau_i,K) = \{(s,t) \mid \tau_i - K < \tau_s \leq \tau_i \text{ and } (w,s) \models \phi,\ t = \min\{u \mid \tau_s < \tau_u \leq \tau_i, (w,u) \models \psi\}\}$

Fig. 1: Formal semantics of SOLOIST

The formal semantics of SOLOIST is trace-based, i.e., defined on timed ω-words over $2^\Pi \times \mathbb{N}$. A timed sequence $\tau = \tau_0\tau_1\ldots$ is an infinite sequence of values $\tau_i \in \mathbb{N}$ satisfying $\tau_i < \tau_{i+1}$, for all $i \geq 0$, i.e., the sequence increases strictly monotonically. A timed ω-word over alphabet 2^Π is a pair (σ, τ) where $\sigma = \sigma_0\sigma_1\ldots$ is an infinite word over 2^Π and τ is a timed sequence. A timed language over 2^Π is a set of timed words over the same alphabet. Notice that there is a distinction between the integer position i in the timed ω-word and the corresponding integer timestamp τ_i. Figure 1 defines the satisfiability relation $(w,i) \models \phi$ for every timed ω-word w, every position $i \geq 0$ and for every SOLOIST formula ϕ.

We remark that the version of SOLOIST presented here is a restriction of the original one in [6]. To simplify the presentation in the next sections, we dropped first-order quantification on finite domains (which was introduced to support data-carrying events) and limited the argument of the $\mathfrak{D}$ modality to only one pair of events. These restrictions are only syntactic sugar and we refer to [6] for the details of the transformations that provide support for them.

SOLOIST can be used to express some of the most common specifications found in service-level agreements (SLAs) of SBAs. Based on our previous study [7], below we list some examples of quantitative specifications found in SLAs, expressed first in English and then in SOLOIST. We refer to generic service operations called `A`, `B`, `C`, `D`, `E`, which correspond to a generic service invocation; each of these operations has a *start* and an *end* event, denoted with the corresponding subscripts. All properties are under the scope

[1] An extended version of the paper, with additional details, can be found at http://arxiv.org/abs/1409.4653.

[2] As detailed in [6], we assume that two subsequent occurrences of the ϕ or ψ events may not happen. This models the behavior of synchronous service invocations.

of an implicit universal temporal quantification as in *"In every process run, ..."*; we assume the time units to be in seconds.

QP1: "The number of invocations of operation A performed within 10 minutes before operation B is invoked is less than or equal to 3". This property is expressed as: $\mathsf{G}(\mathtt{B}_{start} \rightarrow \mathfrak{C}^{600}_{\leq 3}(\mathtt{A}_{end}))$.

QP2: "The average response time of operation C is always less than 5 seconds within any 15 minute time window". This property is expressed as: $\mathsf{G}(\mathfrak{D}^{900}_{<5}(\mathtt{C}_{start}, \mathtt{C}_{end},))$.

QP3: "The maximum number of invocations of operation D is restricted to at most 2 per minute within 10 minutes before operation E is invoked". This property is expressed as: $\mathsf{G}(\mathtt{E}_{start} \rightarrow \mathfrak{M}^{600,60}_{\leq 2}(\mathtt{D}_{end}))$.

B. CLTLB($\mathscr{D}$)

CLTLB($\mathscr{D}$) [10] is an extension of PLTLB (Propositional Linear Temporal Logic with both past and future modalities) [13] augmented with atomic formulae built over a constraint system $\mathscr{D}$. In practice, CLTLB($\mathscr{D}$) defines a set of *variables* C and *arithmetical constraints* over a constraint system $\mathscr{D}$; in our case, $\mathscr{D}$ is the structure $(\mathbb{Z}, =, (<_d)_{d\in\mathbb{Z}})$. For this particular combination, decidability of CLTLB($\mathscr{D}$) has been proven in [14]. Each $<_d$ is a binary relation defined as $x <_d y \Leftrightarrow x < y+d$, hence, for example, the notation $x = y+d$ is an abbreviation for $y <_{1-d} x \wedge x <_{d+1} y$. Variables (henceforth called *counters*) receive a separate evaluation at each time instant. In addition to the standard PLTLB temporal operators "*Since*" and "*Until*", CLTLB($\mathscr{D}$) introduces the new construct of *arithmetic temporal term*, defined as $\alpha := c \mid x \mid \mathsf{Y}(x) \mid \mathsf{X}(x)$, where $c \in \mathbb{Z}$ is a constant, $x \in C$ is a counter and Y and X are temporal operators applied to counters. These temporal operators for counters return the value of the counter in the previous and in the next time instant, respectively. Note that we use a syntactically sugared version of PLTLB using metric temporal operators over time intervals, such as U_I. Since time is discrete, they are just a convenient shorthand [15]. The syntax of CLTLB($\mathscr{D}$) is the following:
$\phi ::= p \mid \alpha \sim \alpha \mid \neg\phi \mid \phi \wedge \phi \mid \phi \mathsf{U}_I \phi \mid \phi \mathsf{S}_I \phi \mid \mathsf{X}\phi \mid \mathsf{Y}\phi$
where p is an atomic proposition, $\sim \in \{=, (<_d)_{d\in\mathbb{Z}}\}$, S_I, U_I, X, Y are the usual "*Since*", "*Until*", "*Next*", and "*Yesterday*" modalities of PLTLB. Additional temporal modalities (like G, "Globally", and W, "Weak Until") can be defined using the usual conventions. An example of a CLTLB($\mathscr{D}$) formula is $\mathsf{G}(\phi \rightarrow \mathsf{X}(y) = y+1)$, which states that whenever ϕ is true, the value of counter y in the next time instant must be incremented of 1 with respect to the value at the current time instant.

CLTLB($\mathscr{D}$) formulae admit finite, ultimately periodic two-part models (π, δ). Function $\pi : \mathbb{N} \rightarrow \mathscr{P}(\Pi)$ associates a subset of the propositions with each time instant, while function $\delta : \mathbb{N} \times C \rightarrow \mathbb{Z}$ defines the value of counters at each time position. Hereafter, this two-part model will be graphically represented as in Fig. 3: the topmost row (above the timeline) represents function π (e.g., $\pi(5) = \{\psi\}$); the rows of integers below the timeline represent function δ, i.e., the values of each counter defined in the model. In the example in the figure there are six counters, as shown on the left: $c_\chi, g_{\phi,\psi}, h_{\phi,\psi}, s_{\phi,\psi}, a_{\phi,\psi}, b_{\phi,\psi}$; the δ function is defined so that we have, for example in correspondence with the sixth time instant (position #5), $\delta(5, g_{\phi,\psi}) = 1$, $\delta(5, h_{\phi,\psi}) = 0$, $\delta(5, s_{\phi,\psi}) = 3$, $\delta(5, a_{\phi,\psi}) = 0$, and $\delta(5, b_{\phi,\psi}) = 3$.

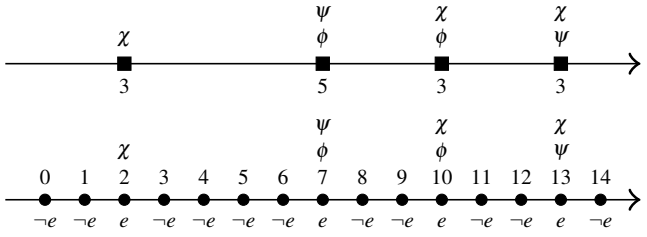

Fig. 2: Mapping a timed ω-word into an ω-word

III. The Translation from SOLOIST to CLTLB($\mathscr{D}$)

The key point in defining the translation from SOLOIST to CLTLB($\mathscr{D}$) is to bridge the gap between the semantics of SOLOIST based on *timed* ω-words, where the temporal information is denoted by an integer time-stamp, and the one of CLTLB($\mathscr{D}$), where the temporal information is implicitly defined by the integer position in an ω-word. The two temporal models can be transformed into each other. Here we are interested in pinpointing, in a CLTLB($\mathscr{D}$) ω-word, only the positions that correspond to actual time-stamps in a SOLOIST timed ω-word. These timestamps correspond to instants where some event actually occurs. To do so, we add to the set Π a special propositional symbol e, which is true in each position corresponding to a "valid" time-stamp in the timed ω-word; a "valid" time-stamp is one where at least an event, represented by a propositional symbol, occurs. An example of this conversion is shown in Fig. 2, where a timed ω-word is depicted in the timeline at the top and its equivalent ω-word corresponds to the timeline at the bottom; notice the special symbols $\neg e$ that hold in positions in the ω-word which do not correspond to a "valid" time-stamp in the timed ω-word. Hereafter, when displaying ω-words, we will omit the symbol e from positions in the timeline, since its presence can be implied by the presence of other propositional symbols in the same position in the timeline.

To define the translation from SOLOIST to CLTLB($\mathscr{D}$) we consider, without loss of expressiveness, only formulae in positive normal form, i.e., where negation may only occur on atoms (see, for example, [15]). First, we extend the syntax of the language by introducing a dual version for each operator in the original syntax, except for the $\mathfrak{C}^K_{\bowtie n}, \mathfrak{U}^{K,h}_{\bowtie n}, \mathfrak{M}^{K,h}_{\bowtie n}, \mathfrak{D}^K_{\bowtie n}$ modalities[3]: the dual of $\wedge$ is $\vee$; the dual of U_I is "*Release*" R_I: $\phi \mathsf{R}_I \psi \equiv \neg(\neg\phi \mathsf{U}_I \neg\psi)$; the dual of S_I is "*Trigger*" T_I: $\phi \mathsf{T}_I \psi \equiv \neg(\neg\phi \mathsf{S}_I \neg\psi)$. A formula is in *positive normal form* if its alphabet is $\{\wedge, \vee, \mathsf{U}_I, \mathsf{R}_I, \mathsf{S}_I, \mathsf{T}_I, \mathfrak{C}^K_{\bowtie n}, \mathfrak{U}^{K,h}_{\bowtie n}, \mathfrak{M}^{K,h}_{\bowtie n}, \mathfrak{D}^K_{\bowtie n}\} \cup \Pi \cup \bar{\Pi}$, where $\bar{\Pi}$ is the set of formulae of the form $\neg p$ for $p \in \Pi$.

We can now illustrate the translation ρ from SOLOIST formulae to CLTLB($\mathscr{D}$). For the propositional ($\neg$, $\wedge$ and $\vee$) and temporal part (U_I, S_I, R_I and T_I) of SOLOIST the translation is straightforward:

$$\rho(p) \equiv p, p \in \Pi; \qquad \rho(\phi \mathsf{U}_I \psi) \equiv (\neg e \vee \rho(\phi)) \mathsf{U}_I (e \wedge \rho(\psi))$$
$$\rho(\neg p) \equiv \neg p, p \in \Pi; \qquad \rho(\phi \mathsf{S}_I \psi) \equiv (\neg e \vee \rho(\phi)) \mathsf{S}_I (e \wedge \rho(\psi))$$

[3] A negation in front of one of the $\mathfrak{C}^K_{\bowtie n}, \mathfrak{U}^{K,h}_{\bowtie n}, \mathfrak{M}^{K,h}_{\bowtie n}, \mathfrak{D}^K_{\bowtie n}$ modalities becomes a negation of the relation denoted by the $\bowtie$ symbol, hence no dual version is needed for them.

$\rho(\phi \wedge \psi) \equiv \rho(\phi) \wedge \rho(\psi)$; $\rho(\phi \mathsf{R}_I \psi) \equiv (e \wedge \rho(\phi)) \mathsf{R}_I (\neg e \vee \rho(\psi))$
$\rho(\phi \vee \psi) \equiv \rho(\phi) \vee \rho(\psi)$; $\rho(\phi \mathsf{T}_I \psi) \equiv (e \wedge \rho(\phi)) \mathsf{T}_I (\neg e \vee \rho(\psi))$

In the rest of this section we focus on the translation of the $\mathfrak{C}^K_{\bowtie n}$ and $\mathfrak{D}^K_{\bowtie n}$ modalities. We omit the translation of the $\mathfrak{U}$ and $\mathfrak{M}$ modalities since they can be expressed in terms of the $\mathfrak{C}$ modality. Specifically, for the $\mathfrak{U}$ modality we exploit the equivalence $\mathfrak{U}^{K,h}_{\bowtie n}(\phi) \equiv \mathfrak{C}^{\lfloor \frac{K}{h} \rfloor \cdot h}_{\bowtie n \cdot \lfloor \frac{K}{h} \rfloor}(\phi)$. As for the $\mathfrak{M}$ modality, the equivalence depends on the $\bowtie$ operator; for example, formula $\mathfrak{M}^{K,h}_{<n}(\phi)$ is equivalent to $\left(\bigwedge_{m=0}^{\lfloor \frac{K}{h} \rfloor - 1} \mathsf{Y}^{m \cdot h}(\ \mathfrak{C}^h_{<n} \phi\right) \wedge \left(\mathsf{Y}^{\lfloor \frac{K}{h} \rfloor \cdot h}(\mathfrak{C}^{(K \bmod h)}_{<n} \phi)\right)$.

A. Translation of the $\mathfrak{C}$ modality

The $\mathfrak{C}$ modality expresses a bound on the number of occurrences of a certain event in a given time window; it comes natural to use the counters available in CLTLB($\mathscr{D}$) for the translation. Indeed, for each sub-formula of the form $\mathfrak{C}^K_{\bowtie n}(\chi)$, we introduce a counter c_χ, constrained by a set of CLTLB($\mathscr{D}$) axioms, detailed below. Informally, these axioms define the value of c_χ such that at each time position it captures the number of occurrences of event χ seen in the past:

A1) $c_\chi = 0$
A2) $\mathsf{G}((e \wedge \chi) \rightarrow \mathsf{X}(c_\chi) = c_\chi + 1)$
A3) $\mathsf{G}((\neg e \vee \neg \chi) \rightarrow \mathsf{X}(c_\chi) = c_\chi)$

Axiom A1 initializes the counter to zero. Axiom A2 states that if there is an occurrence of a valid event χ, (denoted by $e \wedge \chi$) the value of the counter c_χ in the next time instant is increased by one with respect to the value at the current time instant. Axiom A3 refers to the opposite situation, when either there is no occurrence of the event χ or the time instant is not valid (i.e., e does not hold in that time instant). In this case, the value of the counter in the next time instant must have the same value as in the current time instant. Both axioms A2 and A3 have to hold at every time instant, so they are in the scope of a *globally* temporal operator.

We can calculate the exact number of occurrences by subtracting the values of the counter at the appropriate time instants; we explain this through the example in Fig. 3, which depicts a short trace of length 21 and the values assumed by the counter c_χ (in the first row) at each time instant, as determined by the axioms. In the example, to evaluate the formula $\mathfrak{C}^K_{>1}(\chi)$ with $K = 11$ at time instant $t = 16$, we subtract from the value of the counter c_χ at time instant $t+1 = 17$ (since we want to consider a possible occurrence of χ at time instant t) the value of the counter at time instant 6 (i.e., $t-(K-1) = 16-(11-1)$, which is 11 time instants in the past with respect to time instant $t+1$); these values are enclosed in the figure with diamond markers. The value resulting from the subtraction $6-1 = 5$ is then compared to the specified bound ($5 > 1$). In symbols, this can be written as $\mathsf{X}(c_\chi) - \mathsf{Y}^{10}(c_\chi) > 1$ evaluated at time instant t. This intuition is captured by the following CLTLB($\mathscr{D}$) formula, which generalizes the translation of a SOLOIST sub-formula of the form $\mathfrak{C}^K_{\bowtie n}(\chi)$:

$$\rho\left(\mathfrak{C}^K_{\bowtie n}(\chi)\right) \equiv \mathsf{X}(c_\chi) - \mathsf{Y}^{K-1}(c_\chi) \bowtie n$$

Notice that the axioms are conjuncted with the resulting translation of the SOLOIST formula, thus effectively constraining the behavior of all the counters of type c_χ.

B. Translation of the $\mathfrak{D}$ modality

The $\mathfrak{D}$ modality expresses a bound on the average distance between the occurrences of pairs of events in a given time window. As anticipated in Sect. II-A, we consider only (sub)formulae of the $\mathfrak{D}$ modality that refer to one pair, like $\mathfrak{D}^K_{\bowtie n}(\phi, \psi)$.

Events, corresponding to atomic propositions in SOLOIST, can occur multiple times in a trace; when we refer to a specific occurrence of an event ϕ at a time instant τ, we denote this as $\phi_{|\tau}$. Clearly, a pair of events (ϕ, ψ) may also have multiple instances in a trace. We call a pair of the form $(\phi_{|i}, \psi_{|j})$ an *instance* if there is an occurrence of event ϕ at time instant i and an occurrence of event ψ at time instant j, with $i < j$. We call such instance *open* at time instant τ if $i \leq \tau < j$. Otherwise, the instance is *closed* at time instant τ. The *distance* of a closed (pair) instance is $j - i$; for an open pair at time instant τ, the distance is $\tau - i$. A time window of length K defined for a $\mathfrak{D}$ modality (sub-)formula evaluated at time instant τ is bounded by the time instants $\tau+1$ and $\tau-K+1$. For a certain trace, we say that a $\mathfrak{D}$ modality (sub-)formula for a pair of events (ϕ, ψ) has a *left-open* pair in the trace if there is an open instance of (ϕ, ψ) at time instant $\tau-K+1$ in the trace; similarly, we say that the (sub-)formula has a *right-open* pair in the trace if there is an open instance of (ϕ, ψ) at time instant $\tau+1$ in the trace. The translation has then to take into account four distinct cases, depending on whether a $\mathfrak{D}$ modality (sub-)formula contains either (left- and/ or right-) open pairs or none.

As done in the case of the $\mathfrak{C}$ modality, the translation is based on CLTLB($\mathscr{D}$) counters. For each sub-formula of the form $\mathfrak{D}^K_{\bowtie n}(\phi, \psi)$, we introduce five counters, namely:

- $g_{\phi,\psi}$: this binary counter assumes value 1 in the time instants following an occurrence of ϕ and it is reset to 0 after an occurrence of ψ. It acts as a flag denoting the time instants during which the event pair instance is open;
- $h_{\phi,\psi}$: in each time instant, this counter contains the number of previously-seen closed pair instances. It is increased after every occurrence of ψ;
- $s_{\phi,\psi}$: at each time instant, the value of this counter corresponds to the sum of distances of all previously occurred pair instances. It is increased at every time instant when either $g_{\phi,\psi} = 1$ holds or ϕ occurs;
- $a_{\phi,\psi}$: this counter keeps track of the sum of the distances of all previously occurred closed pair instances;
- $b_{\phi,\psi}$: this counter has the values that will be assumed by counter $s_{\phi,\psi}$ at the next occurrence of ψ (more details below).

Counters $a_{\phi,\psi}$, $b_{\phi,\psi}$, and $h_{\phi,\psi}$ are directly used in the translation of the $\mathfrak{D}$ modality (sub-)formulae, while counters $g_{\phi,\psi}$ and $s_{\phi,\psi}$ are helper counters, used to determine the values of the other counters. These five counters are constrained by the following axioms:

A4) $g_{\phi,\psi} = 0 \ \wedge \ h_{\phi,\psi} = 0 \ \wedge \ a_{\phi,\psi} = 0 \ \wedge \ s_{\phi,\psi} = 0$
A5) $(\mathsf{X}(b_{\phi,\psi}) = b_{\phi,\psi}) \mathsf{W} (e \wedge \psi)$

A6) $\mathsf{G}\ ((e \wedge \phi \wedge \neg\psi) \rightarrow (\mathsf{X}(g_{\phi,\psi}) = 1\ \wedge \mathsf{X}(s_{\phi,\psi}) = s_{\phi,\psi} + 1\ \wedge \mathsf{X}(h_{\phi,\psi}) = h_{\phi,\psi}\ \wedge \mathsf{X}(a_{\phi,\psi}) = a_{\phi,\psi}))$

A7) $\mathsf{G}\ ((e \wedge \psi \wedge \neg\phi) \rightarrow (\mathsf{X}(g_{\phi,\psi}) = 0\ \wedge \mathsf{X}(h_{\phi,\psi}) = h_{\phi,\psi} + 1\ \wedge\ \mathsf{X}(a_{\phi,\psi}) = s_{\phi,\psi}\ \wedge \mathsf{X}(s_{\phi,\psi}) = s_{\phi,\psi} \wedge b_{\phi,\psi} = s_{\phi,\psi}\ \wedge\ \mathsf{X}((\mathsf{X}(b_{\phi,\psi}) = b_{\phi,\psi})\mathsf{W}(e \wedge \psi))))$

A8) $\mathsf{G}\ ((\neg e \vee (\neg\phi \wedge \neg\psi)) \rightarrow (\mathsf{X}(g_{\phi,\psi}) = g_{\phi,\psi}\ \wedge\ \mathsf{X}(h_{\phi,\psi}) = h_{\phi,\psi} \wedge\ \mathsf{X}(a_{\phi,\psi}) = a_{\phi,\psi}\ \wedge\ (g_{\phi,\psi} = 1 \rightarrow \mathsf{X}(s_{\phi,\psi}) = s_{\phi,\psi} + 1)\ \wedge\ (g_{\phi,\psi} = 0 \rightarrow \mathsf{X}(s_{\phi,\psi}) = s_{\phi,\psi})))$

A9) $\mathsf{G}\ ((e \wedge \phi \wedge \psi) \rightarrow (\mathsf{X}(g_{\phi,\psi}) = g_{\phi,\psi} \wedge \mathsf{X}(h_{\phi,\psi}) = h_{\phi,\psi} + 1\ \wedge\ \mathsf{X}(a_{\phi,\psi}) = a_{\phi,\psi} \wedge \mathsf{X}(s_{\phi,\psi}) = s_{\phi,\psi}\ \wedge\ \mathsf{X}((\mathsf{X}(b_{\phi,\psi}) = b_{\phi,\psi})\mathsf{W}(e \wedge \psi))$

Axiom A4 initializes all counters except counter $b_{\phi,\psi}$, which will assume values determined by counter $s_{\phi,\psi}$. Axiom A5 states that the value of counter $b_{\phi,\psi}$ will stay the same in all the time instants until the first occurrence of ψ. Notice that we use the W modality ("weak until"), to deal with traces without occurrences of ψ. Axiom A6 determines the next time instant value of the following counters, upon occurrence of a ϕ and absence of a ψ event (denoted by $e \wedge \phi \wedge \neg\psi$): counter $g_{\phi,\psi}$ is set to 1; counter $s_{\phi,\psi}$ is incremented by 1; counters $h_{\phi,\psi}$ and $a_{\phi,\psi}$ are constrained not to change in the next time instant. Axiom A7 determines how the counters are updated when a ψ event occurs and a ϕ event does not: counter $g_{\phi,\psi}$ is set to 0; counters $b_{\phi,\psi}$, $\mathsf{X}a_{\phi,\psi}$, and $\mathsf{X}s_{\phi,\psi}$ are set to be equal to $s_{\phi,\psi}$. Moreover, a formula equivalent to axiom A5 holds in the next time instant, forcing the value of $b_{\phi,\psi}$ to stay the same in all the following time instants until the next occurrence of ψ. Axiom A8 covers the cases either when there are no valid events or when neither ϕ nor ψ occur. In these cases the values of counters $g_{\phi,\psi}$, $h_{\phi,\psi}$, and $a_{\phi,\psi}$ are constrained to stay the same, while counter $b_{\phi,\psi}$ is unconstrained. As for counter $s_{\phi,\psi}$, we need to distinguish two separate cases: when the pair instance is open (denoted by $g_{\phi,\psi} = 1$), counter $s_{\phi,\psi}$ is incremented by 1, otherwise it stays the same. Axiom A9 handles the case when both events ϕ and ψ hold, by incrementing counter $h_{\phi,\psi}$ by 1 and constraining the value of counter $b_{\phi,\psi}$ in the same way like axiom A7. The values of the other counters are constrained to stay the same.

As said above, the $b_{\phi,\psi}$ counter keeps the values that will be assumed by counter $s_{\phi,\psi}$ at the next occurrence of ψ. The value assumed by both counters $a_{\phi,\psi}$ and $b_{\phi,\psi}$ originates from counter $s_{\phi,\psi}$, as enforced by axiom A7. Axioms A6 and A8 make sure the value of $s_{\phi,\psi}$ is propagated in the future via counter $a_{\phi,\psi}$, while axiom A7 enables the propagation of this value in the past via counter $b_{\phi,\psi}$. We elaborate this through an example: Fig. 3 represents a short trace with event ψ occurring at time instants 5, 14, and 19. Axiom A5 enforces equality between successive values of counter $b_{\phi,\psi}$ at adjacent

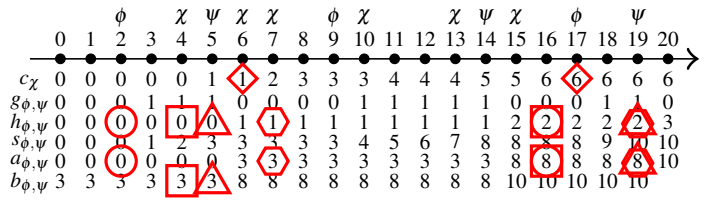

Fig. 3: Sample trace showing the counters used for the translation of the $\mathfrak{C}$ and $\mathfrak{D}$ modalities

time instants until the first occurrence of ψ (time instants 0–5). Additional equalities (of the same type) on the values of counter $b_{\phi,\psi}$ are enforced by axiom A7 (time instants 6–14 and 15–19). The same axiom also determines equality between the values of the $s_{\phi,\psi}$ and $b_{\phi,\psi}$ counters upon an occurrence of ψ (time instants 5, 14 and 19).

The translation $\rho\left(\mathfrak{D}^K_{\bowtie n}(\phi,\psi)\right)$ is defined as:

$$\texttt{if}^4\ (\mathsf{Y}^{K-1}(g_{\phi,\psi}) = 1)\quad \texttt{then}\ \left(\frac{\mathsf{X}(a_{\phi,\psi}) - \mathsf{Y}^{K-1}(b_{\phi,\psi})}{\mathsf{X}(h_{\phi,\psi}) - \mathsf{Y}^{K-1}(h_{\phi,\psi}) - 1} \bowtie n \wedge Z_1\right)$$
$$\texttt{else}\ \left(\frac{\mathsf{X}(a_{\phi,\psi}) - \mathsf{Y}^{K-1}(a_{\phi,\psi})}{\mathsf{X}(h_{\phi,\psi}) - \mathsf{Y}^{K-1}(h_{\phi,\psi})} \bowtie n \wedge Z_2\right)$$

The condition $\mathsf{Y}^K(g_{\phi,\psi}) = 1$ checks whether the time window contains an open pair instance on its left bound. Since the semantics of the $\mathfrak{D}$ modality considers only closed pairs within the time window to compute the average distance, open pairs must be ignored both on the left and on the right bound of the time window. There is no need to differentiate between the cases when there is a right-open pair, since counter $a_{\phi,\psi}$ only considers distances between closed pair instances. The numerator of the fraction in both the `then` and `else` branches denotes the total distance, while the denominator corresponds to the number of pair instances considered for computing the total distance. Propositions Z_1 and Z_2 are respectively $\mathsf{X}(h_{\phi,\psi}) - \mathsf{Y}^{K-1}(h_{\phi,\psi}) \neq 1$ and $\mathsf{X}(h_{\phi,\psi}) - \mathsf{Y}^{K-1}(h_{\phi,\psi}) \neq 0$; due to these disjuncts the $\mathfrak{D}$ modality evaluates to true when there are no closed pairs in the time window K. Axioms A4, A5, A6, A7, A8, A9 are conjuncted with the resulting translation and added as constraints that hold at the initial time instant of the trace.

An example of the use of counters to evaluate a formula with the $\mathfrak{D}$ modality is shown in Fig. 3, which depicts a simple trace and the values assumed by the counters $g_{\phi,\psi}$, $h_{\phi,\psi}$, $s_{\phi,\psi}$, $a_{\phi,\psi}$, and $b_{\phi,\psi}$ at each time instant, as determined by the axioms. We notice that there are three instances of the (ϕ,ψ) pair. If we evaluate the formula $\mathfrak{D}^{14}_{\bowtie n}(\phi,\psi)$ at time instant 15, the two pair instances $(\phi_{|2},\psi_{|5})$ and $(\phi_{|9},\psi_{|14})$, considered to compute the average distance, are closed. The left-hand side (lhs) of the comparison operator ($\bowtie$) is evaluated using the values of counters $a_{\phi,\psi}$ and $h_{\phi,\psi}$ at time instants 16 and 2 (enclosed in a circle in the figure), resulting in $\frac{8}{2} = 4$. When the same formula is evaluated at time instant 18, the portion of the trace considered contains both a left-open $(\phi_{|2},\psi_{|5})$ pair and a right-open $(\phi_{|17},\psi_{|19})$ one. The lhs of the comparison operator is evaluated using the values of counters $a_{\phi,\psi}$, $b_{\phi,\psi}$, and $h_{\phi,\psi}$ at time instants 19 and 5 (enclosed in a triangle in the figure); its value is $\frac{5}{1} = 5$. Now consider the formula $\mathfrak{D}^{12}_{\bowtie n}(\phi,\psi)$. When evaluated at time instant 15, it has a left-open pair $(\phi_{|2},\psi_{|5})$. The values of the counters $a_{\phi,\psi}$, $b_{\phi,\psi}$, and $h_{\phi,\psi}$ considered to compute the lhs of the comparison operator are those at time instants 16 and 4 (enclosed in a square in the figure); the lhs evaluates to $\frac{5}{1} = 5$. If the same formula is evaluated at time instant 18, we find only a right-open pair $(\phi_{|17},\psi_{|19})$. The lhs of the comparison operator is evaluated using the value of counters $a_{\phi,\psi}$ and $h_{\phi,\psi}$ considered at time instants 19 and 7 (enclosed in a hexagon in the figure); its value is $\frac{5}{1} = 5$.

[4] "`if` A `then` B `else` C" can be written as $(A \wedge B) \vee (\neg A \wedge C)$

C. Complexity of the translation

The translation function ρ, for the atomic propositions, the temporal modalities and all the aggregate ones, introduces a fixed-length formula; notice that subformulae occurring in aggregate modalities are restricted to be atomic. In the worst case, our translation is linear in the size of the input formula. We remark that we use a direct encoding of the exponent K in formulae of the form Y^K or X^K, both in the case of arithmetical temporal terms and of boolean formulae. The direct encoding of the exponent allows us to avoid expanding it into nested Y or X formulae.

IV. Implementation

The translation described in the previous section has been implemented in a tool [16]; this tool acts as a front-end for translating SOLOIST formulae into the input format of the ZOT verification toolset [11]. ZOT supports satisfiability checking of CLTLB($\mathscr{D}$) formulae by means of SMT solvers. A plugin-based architecture makes it easy to extend ZOT to support more expressive languages using CLTLB($\mathscr{D}$) as a core, and to output code for the different dialects of various SMT solvers. We implemented the support for SOLOIST as a ZOT plugin written in Common Lisp.

We now give a rundown of the translation steps applied to an example, to provide a glimpse of the implementation of our SMT-based trace checking algorithm. These steps and the example are also sketched in Fig. 4 where: the top row shows (a fragment of) the example input trace and the SOLOIST formula to verify on the trace; the middle row shows how the input trace is transformed from timed ω-word to ω-word, the translation of the input formula and the definition of the counter constraints as described in Sect. III; the bottom row shows how the trace, the input formula, and the counter constraints are translated into the input language of the SMT solver.

Let us consider the problem of performing trace checking of the formula $\phi \equiv \mathfrak{C}^5_{<3}(p)$ over the trace H of length 7 depicted in Fig. 4; the formula is evaluated at time instant 5. As described in Sect. III-A, our plugin translates the SOLOIST formula ϕ into CLTLB($\mathscr{D}$) as $\rho(\phi) \equiv \mathsf{X}(c_p) - \mathsf{Y}^4(c_p) < 3$, where c_p is a counter. The behavior of this counter is constrained by the conjunction of axioms A1, A2, and A3, defined as $\mathscr{C}_{c_p} \equiv (c_p = 0) \wedge \mathsf{G}((e \wedge p) \rightarrow \mathsf{X}(c_p) = c_p + 1) \wedge \mathsf{G}((\neg e \vee \neg p) \rightarrow \mathsf{X}(c_p) = c_p)$. The next step is to invoke ZOT to translate the input formula and the counter constraints into the input language of the SMT solver. First, ZOT parses the formula and assigns a special proposition to each sub-formula in the input formula; similarly, it also assigns an arithmetic proposition to each arithmetical temporal term in the input formula. For example, as shown in Fig. 4, the arithmetic propositions a and b correspond, respectively, to the arithmetical temporal terms $\mathsf{X}(c_p)$ and $\mathsf{Y}^4(c_p)$; c is an arithmetical proposition holding the value of the $\mathsf{X}(c_p) - \mathsf{Y}^4(c_p)$ arithmetic temporal term; proposition d corresponds to the entire input formula. The values of these auxiliary propositions are defined in each time instant $i = 0 \ldots 6$, according to their semantics. The trace H is also encoded in the input language of the SMT solver and provided to it as an assumption. The SMT solver is then fed with the translation, performed by ZOT, of the CLTLB($\mathscr{D}$) formula $\neg(\mathsf{X}^5(\rho(\phi))) \wedge \mathscr{C}_{c_p}$. Notice that the formula ϕ is negated; hence, it is satisfied by trace H if the SMT solver returns *unsat*. The exponent 5 in the term $\mathsf{X}^5(\rho(\phi))$ is determined by the evaluation of the formula fixed at time instant 5. The details of the translation from CLTLB($\mathscr{D}$) to the input language of the SMT solver (as sketched in the bottom row of Fig. 4) have been omitted since they are out of the scope of this work; for them, we refer the reader to [11].

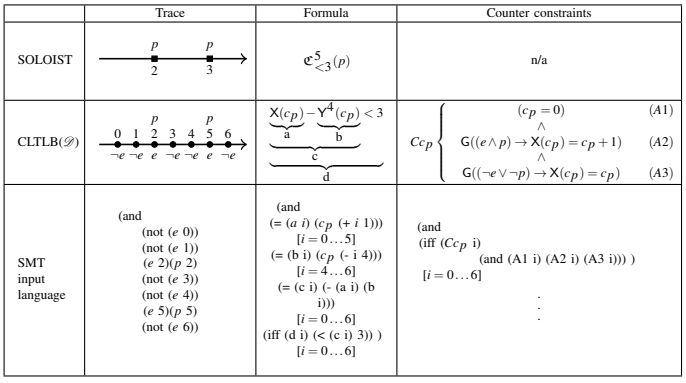

Fig. 4: Example of the translation from SOLOIST to CLTLB($\mathscr{D}$) and then to the input language of the SMT solver

V. Evaluation

We evaluated the effectiveness of our approach by investigating the following research questions:

- RQ1: *How does the proposed approach scale with respect to the various parameters (e.g., the length of the trace, the length of the time window K) involved in SOLOIST trace checking?* (Sect. V-A)
- RQ2: *How does the proposed trace checking procedure for SOLOIST based on CLTLB($\mathscr{D}$) compare with the procedure based on QF-EUFIDL [12]?* (Sect. V-B)

Since there is no consolidated benchmark for service-based applications (for which SOLOIST was tailored), we decided to evaluate our approach using synthesized traces. All traces were synthesized with the Process Log Generator (PLG) tool [17], starting from a model of a realistic service composition (the "Order Booking" business process distributed with the Oracle SOA Suite), comprising 37 activities. This model was defined by specifying the workflow structure, the duration of each activity invoking an external service[5], the branching probabilities (to simulate loops), and the error rates. For each run of the trace checker, we recorded the memory usage and the SMT verification time. The evaluation was performed on a PC equipped with a 2.0GHz Intel Core i7-2630QM processor, running GNU/Linux Ubuntu 12.10 64bit, with 2GB RAM allocated for the verification tool. We used the *Z3* [18] SMT solver v. 4.3.1.

A. RQ1: Scalability of the approach

To investigate RQ1, we considered the following parameters:

[5]Other activities were given 0 as duration.

Trace length. It represents the length of the synthesized trace and the bound given to the SMT solver. The length of each synthesized trace depends on the duration of the activities invoking an external service as well as on the branching probabilities of the loop(s) in the process.

Length of the time window. It is used in the aggregate modalities; it corresponds to the K parameter.

Bound of the comparison operator. It is used in the aggregate modalities; it corresponds to the n parameter.

We present only the results of the evaluation done for the $\mathfrak{C}$ and $\mathfrak{D}$ modalities, since they are the keystones of the translation. We synthesized 20000 different traces, of variable length between 100 and 2000. We checked the following properties on them: $\mathfrak{C}^{100}_{>30}(p)$, and $\mathfrak{D}^{100}_{>30}(p,q)$, with propositions p and q corresponding to the start and end events of a service invocation of the process. The results of executing trace checking for each of these two properties on the synthesized traces are shown in the top plot in Fig. 5; the plot shows the time taken by the trace checking procedure for checking both the $\mathfrak{C}$ properties (shown in black) and the $\mathfrak{D}$ one (shown in gray). Each point in the plot represents an average value of 10 trace check runs on traces of the same length. The plots provide an intuition of the growth rate of the resources usage with respect to the length of the input trace. The memory usage for the respective properties yields a very similar plot; we omitted it for space reasons. The memory dedicated for the evaluation of properties with the $\mathfrak{C}$ modality was exhausted at 2200 time instances, requiring 2.1GB of memory and 40 seconds to solve. For the evaluation of the properties with the $\mathfrak{D}$ modality, the maximum number of time instances manageable before exhausting the preset memory limit was 2000. The lower value with respect to the $\mathfrak{C}$ modality is due to the linear multivariate constraints introduced in the translation of the $\mathfrak{D}$ modality; these constraints are harder to solve than the univariate one used for the $\mathfrak{C}$ modality.

As for the scalability with respect to the other parameters, namely the length of the time window K and the bound of the comparison operator n, we notice that they do not affect the resource usage, and only introduce some non-deterministic noise in the SMT solver time. We evaluated the time and memory usage with respect to the variation of each of these two parameters when checking properties over a synthesized trace of length fixed to 1000. We omit the corresponding plots, since they show a constant value of time/memory with respect to any value of K and n in the properties..

B. RQ2: Comparison with the QF-EUFIDL-based encoding

To investigate RQ2, we compared our approach with previous work for trace checking of SOLOIST [12], based on an encoding of SOLOIST properties into formulae of QF-EUFIDL, the theory of quantifier-free integer difference logic with uninterpreted function and predicate symbols. This encoding was tailored for sparse traces, i.e., traces in which the number of time instants when events occur is very low with respect to the length of the trace.

The comparison focuses on how the two approaches deal with traces of various sparseness degrees, where sparseness is defined as the ratio between the number of time instants in the trace where events occur and the total time the trace spans over. We compared the performance of the two approaches by classifying the generated traces into seven groups with 100%, 50%, 33%, 25%, 20%, 16.6%, and 14.3% of sparseness, respectively. We reevaluated the approach from [12] on traces from each group and compared time and memory requirements of both approaches. As shown in the bottom plot in Fig. 5, the approach presented in this paper is more efficient when the degree of sparseness of input traces is 25% or higher. The black line in the plot shows the performance of our approach, while the seven gray lines show our reevaluation of the approach based on QF-EUFIDL, applied to traces from the seven groups aforementioned.

VI. Related Work

This work lies in the wider area of research on verification of SBAs; we refer the reader to various surveys [2], [3], [4], [5], illustrating approaches both for design-time and for run-time verification of functional and QoS properties. In the rest of this section we focus on existing work on trace checking and verification of quantitative properties specified in languages similar to SOLOIST. For a detailed discussion on SOLOIST and related specification languages see [6].

Finkbeiner et al. [19] describe an approach to collect statistics over run-time executions. They extend LTL to return values from a trace and use them to compute aggregate properties of the trace. However, the specification language they use to describe the statistics to collect provides only limited support for timing information. For example, compared to SOLOIST, it cannot express properties on a certain subset of an execution trace. Furthermore, their evaluation algorithm relies on the formalism of algebraic alternating automata. These automata are manually built from the specification; thus making frequent changes to the property error-prone.

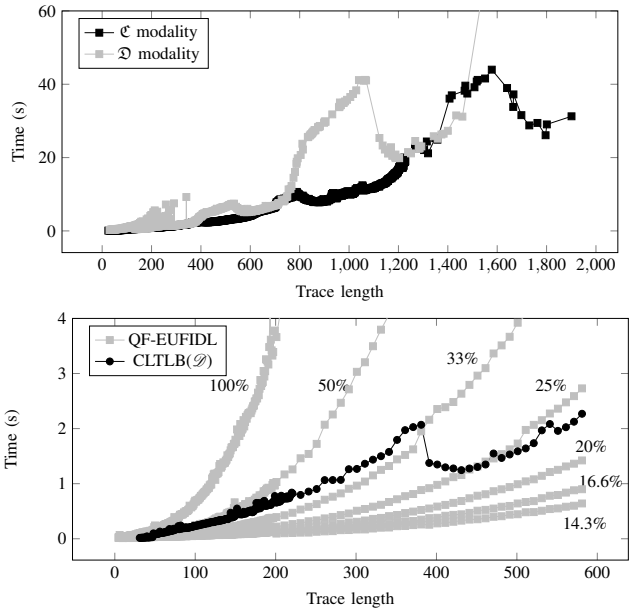

Fig. 5: Scalability with respect to the trace length (top) and comparison between QF-EUFIDL- and CLTLB($\mathscr{D}$)-based encodings (bottom)

In reference [20] authors define an extension of metric first-order temporal logic (MFOTL) which supports aggregation. This language is very similar to SOLOIST with a general definition that supports any aggregate operator that can be defined as a mapping from multisets to $\mathbb{Q} \cup \{\bot\}$. The language can express aggregate properties over the values of the parameters of relations, while SOLOIST expresses aggregate properties on the occurrences of relations in the temporal first-order structure.

The trace checking approach presented in [21] exploits a Map-Reduce framework to validate properties of traces written in LTL. This work mainly focuses on recasting the trace checking problem into a Map-Reduce framework, by distributing (sub)trace validation tasks over many parallel sites.

In reference [22], authors introduce a specification language $PTLTL^{FO}$ (past time linear temporal logic with first-order (guarded) quantifiers) with a counting quantifier. It is used for expressing policies that can categorize the behavioral patterns of a user based on its transaction history. The counting quantifier counts the occurrences of an event from the beginning of the trace until the position of evaluation. The difference with the $\mathfrak{C}$ modality of SOLOIST is that there is no timing information: this means one cannot specify the exact part of the trace the modality should consider.

In reference [23], de Alfaro proposes pTL and pTL* as probabilistic extensions of CTL and CTL*. These new languages include a new modality $\mathfrak{D}$ that expresses the bound on the average time between events. This is achieved by using an instrumentation clock that keeps track of the elapsed time from the beginning of the computation until the first occurrence of a specified event. To this end, the extended pTL formulae are evaluated on an instrumented timed probabilistic Markov decision process. Notice that the $\mathfrak{D}$ modality used in [23] differs from the one we introduced here, since it computes the time passed before the first occurrence of an event, averaged over the different computations of the underlying Markov decision process.

VII. Conclusion and Future Work

The interactions among the various services participating in a composite SBA and the provisioning of such services can be characterized by precise specification patterns [7]. The SOLOIST language was developed [6] to express these patterns, which involve aggregate operations on events occurring in a given time window. In this paper, we propose an SMT-based offline trace checking procedure for SOLOIST. This approach exploits a translation of SOLOIST into CLTLB($\mathscr{D}$), a variant of linear temporal logic that supports counter variables. We assess the scalability of the approach with respect to the various parameters involved in SOLOIST trace checking, and we also compare it with previous work.

The use of SOLOIST in the context of practical verification activities is the goal of further on-going research and we intend to validate our proposal in realistic scenarios, in collaboration with industrial partners. After further improvements to the translation, we also plan to move from offline trace checking to run-time verification, integrating ZOT and the SOLOIST plugin into a run-time monitoring framework for SBAs.

References

[1] N. Josuttis, *SOA in Practice: The Art of Distributed System Design*. O'Reilly Media, Inc., 2007.

[2] M. Bozkurt, M. Harman, and Y. Hassoun, "Testing & verification in service-oriented architecture: A survey," *Softw. Test. Verif. Reliab.*, 2012.

[3] G. Salaün, "Analysis and verification of service interaction protocols - a brief survey," in *Proc. of TAV-WEB 2010*, ser. EPTCS, vol. 35, 2010, pp. 75–86.

[4] G. Canfora and M. Di Penta, "Service oriented architectures testing: a survey," in *ISSSE 2006–2008*, ser. LNCS. Springer, 2009, vol. 5413, pp. 78–105.

[5] L. Baresi and E. Di Nitto, Eds., *Test and Analysis of Web Services*. Springer, 2007.

[6] D. Bianculli, C. Ghezzi, and P. San Pietro, "The tale of SOLOIST: a specification language for service compositions interactions," in *Proceedings of FACS 2012*, vol. 7684. Springer, September 2012, pp. 55–72.

[7] D. Bianculli, C. Ghezzi, C. Pautasso, and P. Senti, "Specification patterns from research to industry: a case study in service-based applications," in *Proceedings of ICSE 2012*. IEEE, June 2012, pp. 968–976.

[8] A. Mrad, S. Ahmed, S. Hallé, and E. Beaudet, "Babeltrace: A collection of transducers for trace validation," in *Proc. of RV 2012*, ser. LNCS, vol. 7687. Springer, 2013, pp. 126–130.

[9] M. Felder and A. Morzenti, "Validating real-time systems by history-checking TRIO specifications," *ACM Trans. Softw. Eng. Methodol.*, vol. 3, no. 4, pp. 308–339, Oct. 1994.

[10] M. M. Bersani, A. Frigeri, A. Morzenti, M. Pradella, M. Rossi, and P. San Pietro, "Constraint ltl satisfiability checking without automata," *CoRR*, vol. abs/1205.0946, 2012.

[11] M. Pradella, A. Morzenti, and P. San Pietro, "Bounded satisfiability checking of metric temporal logic specifications," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 3, pp. 20:1–20:54, Jul. 2013.

[12] M. M. Bersani, D. Bianculli, C. Ghezzi, S. Krstić, and P. San Pietro, "SMT-based checking of SOLOIST over sparse traces," in *Proceedings of FASE 2014*, vol. 8411. Springer, April 2014, pp. 276–290.

[13] O. Lichtenstein, A. Pnueli, and L. Zuck, "The glory of the past," in *Proc. of Logics of Programs*, ser. LNCS, vol. 193. Springer, 1985, pp. 196–218.

[14] S. Demri and D. D'Souza, "An automata-theoretic approach to constraint LTL," *Inf. Comput.*, vol. 205, no. 3, pp. 380–415, 2007.

[15] M. Pradella, A. Morzenti, and P. San Pietro, "The symmetry of the past and of the future: bi-infinite time in the verification of temporal properties," in *Proc. of ESEC-FSE '07*. ACM, 2007, pp. 312–320.

[16] S. Krstić, "SOLOIST Translator," https://bitbucket.org/krle/soloist-translator, 2013.

[17] A. Burattin and A. Sperduti, "PLG: A framework for the generation of business process models and their execution logs," in *Business Process Management Workshops*, ser. LNBIP, vol. 66. Springer, 2011, pp. 214–219.

[18] L. M. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *Proc. of TACAS 2008*, ser. LNCS, vol. 4963. Springer, 2008, pp. 337–340.

[19] B. Finkbeiner, S. Sankaranarayanan, and H. Sipma, "Collecting statistics over runtime executions," *Formal Methods in System Design*, vol. 27, pp. 253–274, 2005.

[20] D. Basin, F. Klaedtke, S. Marinovic, and E. Zălinescu, "Monitoring of temporal first-order properties with aggregations," in *Proc. of RV'13*, ser. LNCS, vol. 8174. Springer, 2013, pp. 40–58.

[21] B. Barre, M. Klein, M. Soucy-Boivin, P.-A. Ollivier, and S. Hallé, "MapReduce for Parallel Trace Validation of LTL Properties," in *Proc. of RV 2012*, ser. LNCS, vol. 7687. Springer, 2013, pp. 184–198.

[22] A. Bauer, R. Goré, and A. Tiu, "A first-order policy language for history-based transaction monitoring," in *Proc. of ICTAC '09*, ser. LNCS, vol. 5684. Springer, 2009, pp. 96–111.

[23] L. de Alfaro, "Temporal logics for the specification of performance and reliability," in *Proc. of STACS'97*, ser. LNCS, vol. 1200. Springer, 1997, pp. 165–176.