From SOLOIST to CLTLB($\mathscr{D}$): Checking Quantitative Properties of Service-based Applications

Domenico Bianculli[1], Carlo Ghezzi[2],
Srđan Krstić[2] and Pierluigi San Pietro[2]
[1]SnT Centre - University of Luxembourg, Luxembourg
[2]DEEPSE group - DEIB - Politecnico di Milano, Milano, Italy
domenico.bianculli@uni.lu, carlo.ghezzi@polimi.it,
srdan.krstic@polimi.it, pierluigi.sanpietro@polimi.it

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.

In this paper we address verification of requirements specifications written in SOLOIST, by presenting a translation of SOLOIST into CLTLB($\mathscr{D}$), a variant of linear temporal logic augmented with atomic formulae built over a constraint system. This translation allows us to efficiently check service execution traces against quantitative temporal properties, involving aggregate operations on events occurring in a given time window. We implemented the translation as a plugin for the ZOT verification toolkit. We detail the results of evaluating the achieved scalability, with respect to several parameters.

1 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 [23]. SBAs are usually defined in terms of service compositions, resulting from the orchestration of several existing services, possibly provided by third-parties, by means of dedicated languages such as BPEL [2]. Developing and operating SBAs involve many stakeholders, including service end-users, the developers and providers of all services defined in an SBA, as well as the service integrators that realize composite services. All these stakeholders may belong to different companies or different divisions within the same company. In both cases, service integrators have the ultimate responsibility for maintaining an adequate level of quality attributes — expressed both 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.

Several verification techniques have been developed and tailored [12, 30, 14, 4] 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). 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 [10], some of the authors performed a field study in the context of enterprise SBAs in collaboration with an industrial partner. The study collected and classified the main specification patterns used to write requirements specifications for SBAs, both in a research [24, 18]) and in an industrial setting. One of the results of this study was 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 events* occurrences in a given time window.

The results of this study prompted the definition of a new specification language, able to express the identified service provisioning patterns. The language, introduced in [11], is called SOLOIST (*SpecificatiOn Language fOr servIce compoSitions inTeractions*) and is a many-sorted first-order metric temporal logic with new temporal modalities that support aggregate operations on events occurring in a given time window.

In the original definition of SOLOIST [11] 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 of verification procedures.

The main contribution of this paper is a verification procedure for SOLOIST based on a new, efficient translation into CLTLB($\mathscr{D}$) [8], an extension of PLTLB (Propositional Linear Temporal Logic with both past and future modalities) augmented with atomic formulae built over a constraint system $\mathscr{D}$. In particular, in this paper we report on our practical experience of performing *trace checking* of service execution traces against requirements expressed in SOLOIST. We use the procedure of bounded satisfiability checking of metric temporal logic specifications supported by ZOT [29], a verification toolset based on SMT-solvers. 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. The results show how the proposed translation scales with respect to the different parameters involved in SOLOIST trace checking, such as the length of the traces and the size of the time window over which the aggregate modalities are applied.

The rest of the paper is organized as follows. Section 2 provides background information on SOLOIST, CLTLB($\mathscr{D}$), and introduces a running example. We present the translation of SOLOIST into CLTLB($\mathscr{D}$) in Section 3. In section 4 we discuss some implementation details. Section 5 reports on the evaluation performed to assess the scalability and applicability of our translation when used in the context of trace checking. Section 6 surveys related work and Section 7 provides some concluding remarks and directions for future work.

2 Preliminaries

2.1 SOLOIST in a Nutshell

Previous work [10] reports the results of a field study on the the specification patterns used to express requirements of SBAs. After analyzing more than 900 requirements specifications extracted both from research papers and industrial data, the authors of the study identified a set of specification patterns specific to service provisioning (in addition to the well-known ones like those defined in [24, 18]). The service provisioning patterns refer to: S1) average response time; S2) counting the number of events; S3) average number of events; S4) maximum number of events; S5) absolute time; S6) unbounded elapsed time; S7) data-awareness. On the basis of these findings, a new specification language, called SOLOIST (*SpecificatiOn Language fOr servIce compoSitions inTeractions*), was introduced in [11]. SOLOIST was in fact designed with the goal of supporting the common specification patterns found for service provisioning; it is a many-sorted first-order metric temporal logic with new temporal modalities that support aggregate operations on events occurring in a given time window.

The syntax of SOLOIST is defined as follow. A signature Σ is a tuple $\langle S; F; P\rangle$ where S is a set of sort symbols; F is a set of function symbols of the form $f\colon s_1 \times \ldots \times s_n \to w$ with $s_1,\ldots,s_n,w \in S$; P is a set of predicate symbols of the form $p\colon s_1 \times \ldots \times s_n$ with $s_1,\ldots,s_n \in S$. For each sort symbol s in S, V_s is the set of variables of sort s. A term t can be either a variable of a certain sort or the application of a function f to terms $t_1,\ldots,t_n$, as in $f(t_1,\ldots,t_n)$. SOLOIST formulae over a signature Σ are given by the following grammar:
$\phi ::= p(t_1,\ldots,t_n) \mid \neg\phi \mid \phi \wedge \phi \mid \exists x\colon \phi \mid \phi \mathsf{U}_I \phi \mid \phi \mathsf{S}_I \phi \mid \mathfrak{C}^K_{\bowtie n}(\phi) \mid \mathfrak{U}^{K,h}_{\bowtie n}(\phi) \mid \mathfrak{M}^{K,h}_{\bowtie n}(\phi) \mid \mathfrak{D}^K_{\bowtie n}\{(\phi,\phi),\ldots,(\phi,\phi)\}$
where $p \in P$; $t_1,\ldots,t_n$ are terms, each one of the proper sort as prescribed by the definition of the predicate symbol to which the term is an argument; x is a variable from some V_s; I is a nonempty interval over $\mathbb{N}$; n,K,h range over $\mathbb{N}$; $\bowtie \in \{<,\leq,\geq,>,=\}$. We restrict the arguments ϕ of the modalities $\mathfrak{C},\mathfrak{U},\mathfrak{M},\mathfrak{D}$ to be *atoms* of the form $p(t_1,\ldots,t_n)$. Moreover, for the $\mathfrak{D}$ modality, we require the elements of each pair (ϕ_i,ψ_i) to correspond to different atoms.

The U_I and S_I modalities have the usual meaning in temporal logic ("*Until*" and its past counterpart "*Since*"), but they are bound to interval I; additional temporal modalities (like G, "*Globally*") can be defined from them using the usual conventions. The $\mathfrak{C}^K_{\bowtie n}(\phi)$ modality states a bound on the number of occurrences of an event ϕ occurred in the previous K time instants; it expresses pattern S2. 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 — aggregated over an observation interval h — of occurrences of an event ϕ, occurred in the previous K time instants; it corresponds to pattern S3 (respectively, S4). The $\mathfrak{D}^K_{\bowtie n}\{(\phi_1,\psi_1),\ldots,(\phi_n,\psi_n)\}$ modality expresses a bound on the average time elapsed between pairs of specific adjacent events (e.g., ϕ_i and ψ_i), occurred in the previous K time instants; it can be used to express pattern S1. Notice that data-awareness (pattern S7) is supported by means of first-order quantification, following the approach proposed in [22]; however, since we assume that the domains over which the quantification ranges are finite, the quantification is mere syntactic sugar, without impact on the decidability of the language.

$$
\begin{array}{lll}
(\bar{\mathscr{D}},\bar{\tau},\sigma,i) \models p(t_1,\ldots,t_n) & \text{iff} & ([\![t_1]\!],\ldots,[\![t_n]\!]) \in p^{\mathscr{D}_i} \\
(\bar{\mathscr{D}},\bar{\tau},\sigma,i) \models \neg\phi & \text{iff} & (\bar{\mathscr{D}},\bar{\tau},\sigma,i) \not\models \phi \\
(\bar{\mathscr{D}},\bar{\tau},\sigma,i) \models \phi \wedge \psi & \text{iff} & (\bar{\mathscr{D}},\bar{\tau},\sigma,i) \models \phi \wedge (\bar{\mathscr{D}},\bar{\tau},\sigma,i) \models \psi \\
(\bar{\mathscr{D}},\bar{\tau},\sigma,i) \models \exists x\colon \phi & \text{iff} & (\bar{\mathscr{D}},\bar{\tau},\sigma[x/d],i) \models \phi \text{for some } d \in s^{\mathscr{D}} (\text{with } x \text{ of sort } s) \\
(\bar{\mathscr{D}},\bar{\tau},\sigma,i) \models \phi \mathsf{S}_I \psi & \text{iff} & \text{for some } j<i, \tau_i-\tau_j \in I, (\bar{\mathscr{D}},\bar{\tau},\sigma,j) \models \psi \\
 & & \text{and for all } k, j<k<i, (\bar{\mathscr{D}},\bar{\tau},\sigma,k) \models \phi \\
(\bar{\mathscr{D}},\bar{\tau},\sigma,i) \models \phi \mathsf{U}_I \psi & \text{iff} & \text{for some } j>i, \tau_j-\tau_i \in I, (\bar{\mathscr{D}},\bar{\tau},\sigma,j) \models \psi \\
 & & \text{and for all } k, i<k<j, (\bar{\mathscr{D}},\bar{\tau},\sigma,k) \models \phi \\
(\bar{\mathscr{D}},\bar{\tau},\sigma,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 \\
(\bar{\mathscr{D}},\bar{\tau},\sigma,i) \models \mathfrak{U}^{K,h}_{\bowtie n}(\phi) & \text{iff} & \dfrac{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 \\
(\bar{\mathscr{D}},\bar{\tau},\sigma,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{given } lb(m) = \max\{\tau_i-K, \tau_i-(m+1)h\} \\
 & & \text{and } rb(m) = \tau_i - mh, \text{ with } \tau_i \geq K \\
(\bar{\mathscr{D}},\bar{\tau},\sigma,i) \models \mathfrak{D}^K_{\bowtie n}\{(\phi_1,\psi_1),\ldots,(\phi_m,\psi_m)\} & \text{iff} & \dfrac{\sum_{j=1}^{m} \sum_{(s,t)\in d(\phi_j,\psi_j,\tau_i,K)} (\tau_t-\tau_s)}{\sum_{j=1}^{m} |d(\phi_j,\psi_j,\tau_i,K)|} \bowtie n \text{ and } \tau_i \geq K
\end{array}
$$

where $c(\tau_a,\tau_b,\phi) = |\{s \mid \tau_a < \tau_s \leq \tau_b \text{ and } (\bar{\mathscr{D}},\bar{\tau},\sigma,s) \models \phi\}|$, and
$d(\phi,\psi,\tau_i,K) = \{(s,t) \mid \tau_i-K < \tau_s \leq \tau_i \text{ and } (\bar{\mathscr{D}},\bar{\tau},\sigma,s) \models \phi, t = \min\{u \mid \tau_s < \tau_u \leq \tau_i, (\bar{\mathscr{D}},\bar{\tau},\sigma,u) \models \psi\}\}$

Figure 1: Formal semantics of SOLOIST

The formal semantics of SOLOIST can be defined as follows. For a signature $\Sigma = \langle S; F; P \rangle$, let $\mathscr{D}$ be a Σ-structure consisting of: a non-empty set $s^{\mathscr{D}}$ for each sort $s \in S$; a function $f^{\mathscr{D}}\colon s_1^{\mathscr{D}} \times \ldots \times s_n^{\mathscr{D}} \to w^{\mathscr{D}}$ for each function symbol $f \in F$; a relation $p^{\mathscr{D}} \subseteq s_1^{\mathscr{D}} \times \ldots \times s_n^{\mathscr{D}}$ for each predicate symbol $p \in P$. A *temporal first-order* structure over Σ is a pair $(\bar{\mathscr{D}},\bar{\tau})$, where $\bar{\mathscr{D}} = \mathscr{D}_0,\mathscr{D}_1,\ldots$ is a sequence of Σ-structures and $\bar{\tau} = \tau_0,\tau_1,\ldots$ is a sequence of natural numbers (i.e., time-stamps), where the sequence $\bar{\tau}$ is monotonically increasing, the sorts of $\bar{\mathscr{D}}$ are constant, and each function symbol has a rigid interpretation. A variable assignment σ is a S-indexed family of functions $\sigma_s\colon V_s \to s^{\mathscr{D}}$ that maps every variable $x \in V_s$ of sort s to an element $\sigma_s(x) \in s^{\mathscr{D}}$. Notation $\sigma[x/d]$ denotes the variable assignment that maps x to d and maps all other variables as σ does. The valuation function $[\![t]\!]$ of term t for a Σ-structure $\mathscr{D}$ is defined inductively as follows: if t is a variable $x \in V_s$, then $[\![t]\!] = \sigma_s(x)$; if t is a term $f(t_1,\ldots,t_n)$ then $[\![t]\!] = f^{\mathscr{D}}([\![t_1]\!]\ldots,[\![t_n]\!])$. Given a temporal structure $(\bar{\mathscr{D}},\bar{\tau})$ over Σ, a variable assignment σ, $\bowtie \in \{<,\leq,\geq,>,=\}$, $i,n,K,h \in \mathbb{N}$, we define the satisfiability relation $(\bar{\mathscr{D}},\bar{\tau},\sigma,i) \models \phi$ for SOLOIST formulae as depicted in Fig. 1. The detailed rationale behind the semantics of the language can be found in [11].

2.2 Example

In this section we show how to express common quantitative properties of an SBA by means of SOLOIST. As an example, we consider an SBA realized as a service composition described in BPEL, depicted in Fig. 2 using the (visually intuitive) notation introduced in [3].

The process *ATMFrontEnd* starts when the *receive* activity `logOn` processes a message from the *SessionManager* service. This starts a customer session: the process verifies whether the customer holds a valid account at the bank, by invoking the `checkAccess` operation of the *BankAccount* service. If

the latter identifies the customer, a loop is started to manage the customer's requests sent via the *User-Interaction* service. The `customerMenu` *pick* activity, contained in the body of the loop, may receive four kinds of possible requests: three of them (`getBalance`, `deposit`, `withdraw`) are forwarded to the corresponding operations of the *BankAccount* service; the `logOff` request terminates the loop, closing the customer session.

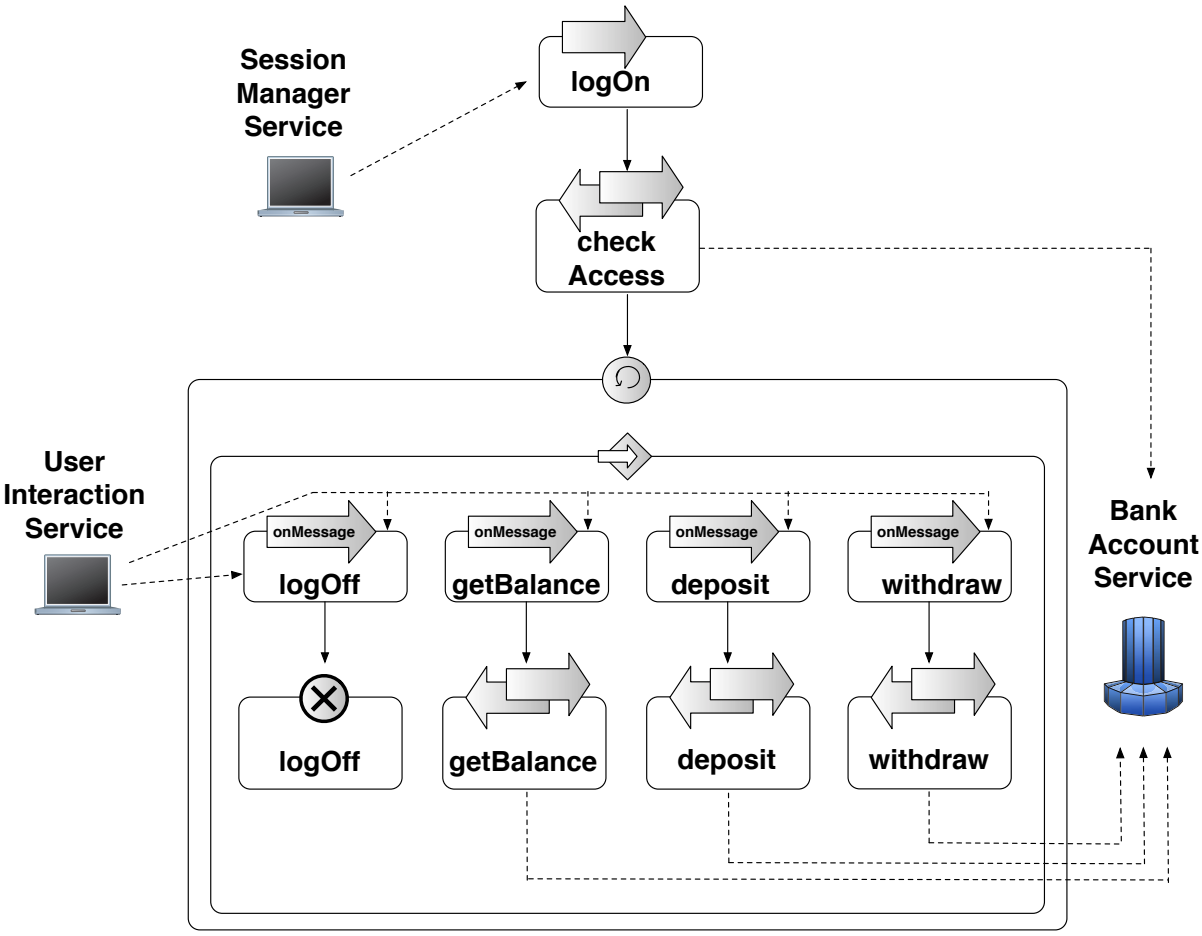


Figure 2: *ATMFrontEnd* business process

To annotate a BPEL process with SOLOIST, we denote the execution of each activity with a predicate symbol. Activities that involve a data exchange (e.g., an *invoke*) are modeled as non-nullary predicates, where the arguments correspond to the variables by which the input (output) messages can be represented. Synchronous *invoke* activities are actually modeled with two predicates, corresponding to the start and the end of the invocation; these are denoted with the "_start" and "_end" suffixes, respectively.

A signature $\Sigma = \langle S; F; P \rangle$ for the example process can be defined as follows. S is the set of XML simple types used across the operations (e.g., integer); F is the set of functions defined by the scripting language used within the process (none in this case); P corresponds to the set of all process activities, as described above.

Below we list some examples of quantitative properties expressed first in natural language and then with SOLOIST; more details on the features of the language are available in [11]. All properties are under the scope of an implicit universal temporal quantification as in *"In every process run, ..."*; we

assume the time units to be in seconds.

QP1: WithdrawalLimit
The number of withdrawal operations performed within 10 minutes before customer logs off is less than or equal to the allowed limit (assumed to be 3, for example). This property is expressed as:
$\mathsf{G}(logOff \rightarrow \mathfrak{C}^{600}_{\leq 3}(withdraw))$.

QP2: CheckAccessAverageResponseTime
The average response time of operation `checkAccess` provided by the *BankAccount* service is always less than 5 seconds within any 15 minute time window. This property is expressed as:
$\mathsf{G}(\mathfrak{D}^{900}_{<5}\{(checkAccess\_start, checkAccess\_end)\})$.

QP3: MaxNumberOfBalanceInquiries
The maximum number of balance inquiries is restricted to at most 2 per minute within 10 minutes before customer session ends. This property is expressed as:
$\mathsf{G}(logOff \rightarrow \mathfrak{M}^{600,60}_{\leq 2}(getBalance))$.

2.3 CLTLB($\mathscr{D}$)

CLTLB($\mathscr{D}$) [8] is an extension of PLTLB (Propositional Linear Temporal Logic with both past and future modalities) [26] 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 which decidability has been proven in [17]; 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 $y <_{-d+1} x \wedge x <_{d+1} y$. Variables (henceforth called *counters*) receive a separate evaluation at each time instant. In addition to the standard PLTLB syntax, 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 that, unlike their PLTLB counterparts, 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 that allows us to express metric temporal operators [28]. The complete syntax of CLTLB($\mathscr{D}$) can be defined as follows:
$\phi ::= p \mid \alpha \sim \alpha \mid \neg\phi \mid \phi \wedge \phi \mid \phi \mathsf{U}_I \phi \mid \phi \mathsf{S}_I \phi \mid \mathsf{X}\phi \mid \mathsf{Y}\phi$
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 is constrained to be increased by 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 set of 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 like in Fig. 5: the topmost row 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 (shown on the left) in each time instant (e.g., $\delta(5, s_\phi) = 3$).

3 From SOLOIST to CLTLB($\mathscr{D}$)

For the translation of SOLOIST into CLTLB($\mathscr{D}$, we consider, without loss of expressiveness, a single-sorted version of the language. We also assume that the domain corresponding to the one sort is finite, so that every first-order quantification can be converted into a conjunction or disjunction of formulae including a finite set Π of new atomic propositions. These simplifications allow us to replace the temporal first-order structure $(\bar{\mathscr{D}}, \bar{\tau})$ and the variable assignment σ, used in the definition of the satisfiability relation of SOLOIST, with *timed* ω-words [1] , i.e., ω-words over $2^{\Pi} \times \mathbb{N}$. An example of *timed* ω-word is depicted in the timeline at the top of Fig. 3. Furthermore, we consider only formulae in positive normal form, i.e., where negation may only occur on atoms (see, for example, [28]). First, we extend the syntax of the language by introducing a dual version for each operator in the original syntax, except for the $\mathfrak{C}^K_{\triangleright\!\!\triangleleft n}, \mathfrak{U}^{K,h}_{\triangleright\!\!\triangleleft n}, \mathfrak{M}^{K,h}_{\triangleright\!\!\triangleleft n}, \mathfrak{D}^K_{\triangleright\!\!\triangleleft n}$ modalities[1]: 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 $\{\wedge, \vee, \mathsf{U}_I, \mathsf{R}_I, \mathsf{S}_I, \mathsf{T}_I, \mathfrak{C}^K_{\triangleright\!\!\triangleleft n}, \mathfrak{U}^{K,h}_{\triangleright\!\!\triangleleft n}, \mathfrak{M}^{K,h}_{\triangleright\!\!\triangleleft n}, \mathfrak{D}^K_{\triangleright\!\!\triangleleft n}\} \cup \Pi \cup \bar{\Pi}$, where $\bar{\Pi}$ is the set of formulae of the form $\neg p$ for $p \in \Pi$.

One 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 a natural 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 the positions in a CLTLB($\mathscr{D}$) ω-word that correspond to time-stamps where events occur in a SOLOIST timed ω-word. 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 (i.e., a time-stamp where at least an event, represented by a propositional symbol, occurs). An example of this conversion is shown in Fig. 3, 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. Hereafter, traces will be depicted as monoinfinite timelines with circles (respectively, squares) representing time instants of ω-words (respectively, timed ω-words). Also, 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 time position in the timeline.

We can now illustrate the translation ρ from single-sorted and quantifier-free 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:

$$
\begin{aligned}
\rho(p) &\equiv p, p \in \Pi \\
\rho(\neg p) &\equiv \neg p, p \in \Pi \\
\rho(\phi \wedge \psi) &\equiv \rho(\phi) \wedge \rho(\psi) \\
\rho(\phi \vee \psi) &\equiv \rho(\phi) \vee \rho(\psi) \\
\rho(\phi \mathsf{U}_I \psi) &\equiv (\neg e \vee \rho(\phi)) \mathsf{U}_I (e \wedge \rho(\psi)) \\
\rho(\phi \mathsf{S}_I \psi) &\equiv (\neg e \vee \rho(\phi)) \mathsf{S}_I (e \wedge \rho(\psi)) \\
\rho(\phi \mathsf{R}_I \psi) &\equiv (e \wedge \rho(\phi)) \mathsf{R}_I (\neg e \vee \rho(\psi)) \\
\rho(\phi \mathsf{T}_I \psi) &\equiv (e \wedge \rho(\phi)) \mathsf{T}_I (\neg e \vee \rho(\psi))
\end{aligned}
$$

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

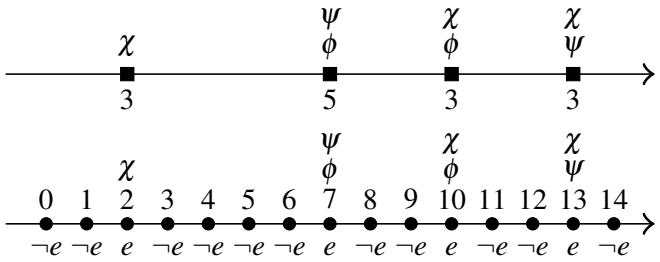


Figure 3: Conversion from timed ω-word to ω-word

In the rest of this section we focus on the translation of the new modalities introduced by SOLOIST, i.e., $\mathfrak{C}^K_{\bowtie n}, \mathfrak{U}^{K,h}_{\bowtie n}$, $\mathfrak{M}^{K,h}_{\bowtie n}, \mathfrak{D}^K_{\bowtie n}$, and then we briefly discuss its complexity.

3.1 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, to translate a SOLOIST formula containing sub-formulae with the $\mathfrak{C}$ modality, we use a counter for the argument of each $\mathfrak{C}$ modality sub-formula.

For each sub-formula of the form $\mathfrak{C}^K_{\bowtie n}(\phi)$, we introduce a counter c_ϕ, constrained by the following CLTLB($\mathscr{D}$) axioms:

A1) $c_\phi = 0$

A2) $\mathsf{G}((e \wedge \phi) \rightarrow \mathsf{X}(c_\phi) = c_\phi + 1)$

A3) $\mathsf{G}((\neg e \vee \neg\phi) \rightarrow \mathsf{X}(c_\phi) = c_\phi)$

Axiom A1 initializes the counter to zero. Axiom A2 is an implication: the antecedent corresponds to the occurrence of a valid event ϕ, denoted by the conjunction $e \wedge \phi$; in the consequent, the value of the counter c_ϕ in the next time instant is constrained to be increased by 1 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 ϕ 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 is constrained to 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.

The translation of a sub-formula of the form $\mathfrak{C}^K_{\bowtie n}(\phi)$ is then defined as:

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

Axioms A1, A2, A3 constrain the behavior of each counter c_ϕ so that c_ϕ holds, at every time instant τ, the number of occurrences of event ϕ in the past. Hence, we can calculate the exact number of occurrences by subtracting the values of the counter at the appropriate time instants; this is exactly what the formula in (1) does. The axioms are conjuncted with the resulting translation, thus effectively constraining the behavior of all the counters of the type c_ϕ.

An example of the use of counters of the type c_ϕ is shown in Fig. 4, which depicts a simple trace and the values assumed by the counter at each time instant, as determined by the axioms. To evaluate the formula $\mathfrak{C}^5_{>1}(\phi)$ at time instant 13 against the sample trace, we subtract the value of the counter c_ϕ

at time instant 9 from the value at time instant 14 (these values are marked with a circle). The resulting value ($10-6=4$) is then compared to the specified bound ($4>1$).

3.2 Translation of the $\mathfrak{U}$ modality

The translation of the $\mathfrak{U}$ modality is defined in terms of the $\mathfrak{C}$ modality; it can then be defined as follow:

$$\rho\left(\mathfrak{U}^{K,h}_{\bowtie n}(\phi)\right) \equiv \rho\left(\mathfrak{C}^{\lfloor \frac{K}{h} \rfloor \cdot h}_{\bowtie n \cdot \lfloor \frac{K}{h} \rfloor}(\phi)\right) \tag{2}$$

This translation indicates that we ignore the tail subinterval of the $\mathfrak{U}$ modality, which is consistent with the SOLOIST semantics [11].

3.3 Translation of the $\mathfrak{M}$ modality

To translate the $\mathfrak{M}$ modality we again rely on the $\mathfrak{C}$ modality. The translation for a formula of the form $\mathfrak{M}^{K,h}_{<n}(\phi)$ is defined as: $\rho\left(\mathfrak{M}^{K,h}_{<n}(\phi)\right) \equiv$

$$\left(\bigwedge_{m=0}^{\lfloor \frac{K}{h} \rfloor - 1} \Upsilon^{m \cdot h}(\rho(\mathfrak{C}^{h}_{<n}\phi))\right) \wedge \left(\Upsilon^{\lfloor \frac{K}{h} \rfloor \cdot h}(\rho(\mathfrak{C}^{(K \bmod h)}_{<n}\phi))\right) \tag{3}$$

For a formula of the form $\mathfrak{M}^{K,h}_{>n}(\phi)$ we have: $\rho\left(\mathfrak{M}^{K,h}_{>n}(\phi)\right) \equiv$

$$\left(\bigvee_{m=0}^{\lfloor \frac{K}{h} \rfloor - 1} \Upsilon^{m \cdot h}\left(\rho\left(\mathfrak{C}^{h}_{>n}\phi\right)\right)\right) \vee \left(\Upsilon^{\lfloor \frac{K}{h} \rfloor \cdot h}\left(\rho\left(\mathfrak{C}^{(K \bmod h)}_{>n}\phi\right)\right)\right) \tag{4}$$

The formulae above decompose the computation of the maximum number of occurrences of the event $(e \wedge \phi)$ by suitably combining constraints on the number of occurrences of the event in each observation interval h within the time window K. The other cases of the operator $\bowtie$ can be defined in a similar way.

3.4 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. For simplicity, 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 $\phi_{|\tau}$. A pair of events (ϕ,ψ) may have multiple instances in a trace. We call a pair like $(\phi_{|i},\psi_{|j})$ *an instance* if there is an

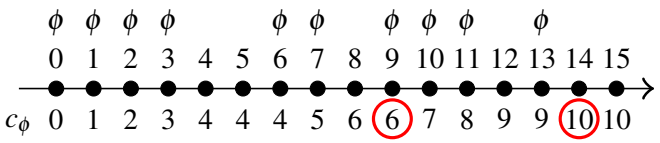


Figure 4: Example of trace for the $\mathfrak{C}$ modality

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 defined for a $\mathfrak{D}$ modality (sub-)formula evaluated at time instant τ is bounded by the time instants $\tau + 1$ and $\tau - K + 1$. Given a trace H, we say that a $\mathfrak{D}$ modality (sub-)formula for a pair of events (ϕ, ψ) has a *left-open* (respectively, *right-open*) pair in H if there is an open instance of (ϕ, ψ) at time instant $\tau - K + 1$ (respectively, $\tau + 1$) in H. Depending on whether a $\mathfrak{D}$ modality (sub-)formula contains either (left- and/ or right-) open pairs or none, there are four distinct cases to take into account for the translation.

For each sub-formula of the form $\mathfrak{D}^K_{\bowtie n}\{(\phi, \psi)\}$, we introduce five counters. The counters are:

- $c_{\phi,\psi}$. This binary counter assumes value 1 in the time instants following an occurrence of ϕ and 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 $c_{\phi,\psi} = 1$ 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 $c_{\phi,\psi}$ and $s_{\phi,\psi}$ are helper counters, used to determine the values of the other counters.

These counters are constrained by the following axioms:

A4) $c_{\phi,\psi} = 0 \wedge h_{\phi,\psi} = 0 \wedge a_{\phi,\psi} = 0 \wedge s_{\phi,\psi} = 0$

A5) $(\mathsf{X}(b_{\phi,\psi}) = b_{\phi,\psi})\mathsf{W}(e \wedge \psi)$

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

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

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

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 consecutive 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 event (denoted by $\phi \wedge e$): counter $c_{\phi,\psi}$ 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 an event ψ occurs: counter $c_{\phi,\psi}$ is set to 0;

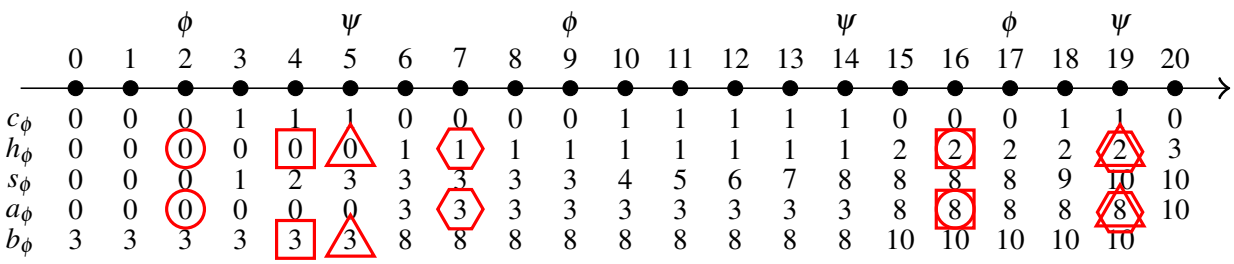


Figure 5: Example of trace for the $\mathfrak{D}$ modality

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, axiom A5 is constrained to hold in the next time instant, forcing value of $b_{\phi,\psi}$ to stay the same in all the consecutive 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 $c_{\phi,\psi}$, $h_{\phi,\psi}$, and $a_{\phi,\psi}$ are constrained to stay the same, counter $b_{\phi,\psi}$ is unconstrained, while for $s_{\phi,\psi}$ we need to distinguish two separate cases. When the pair instance is open (denoted by $c_{\phi,\psi} = 1$), counter $s_{\phi,\psi}$ is incremented by 1, otherwise it stays 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 axioms A5 and A7 enable the propagation of this value in the past via counter $b_{\phi,\psi}$. We elaborate this through an example: Fig. 5 represents a short trace of length 21 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 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 $\rho(\mathfrak{D}^K_{\bowtie n}\{(\phi,\psi)\})$ is defined as:

$$\begin{aligned}\rho\left(\mathfrak{D}^K_{\bowtie n}\{(\phi,\psi)\}\right) \equiv\ & \texttt{if } (\mathsf{Y}^{K-1}(c_{\phi,\psi}) = 1) \\ & \texttt{then } \frac{\mathsf{X}(a_{\phi,\psi}) - \mathsf{Y}^{K-1}(b_{\phi,\psi})}{\mathsf{X}(h_{\phi,\psi}) - \mathsf{Y}^{K-1}(h_{\phi,\psi}) - 1} \bowtie n \\ & \texttt{else } \frac{\mathsf{X}(a_{\phi,\psi}) - \mathsf{Y}^{K-1}(a_{\phi,\psi})}{\mathsf{X}(h_{\phi,\psi}) - \mathsf{Y}^{K-1}(h_{\phi,\psi})} \bowtie n\end{aligned} \tag{5}$$

The condition $\mathsf{Y}^K(c_{\phi,\psi}) = 1$ checks whether the time window contains an open pair instance on its left bound. Notice that the semantics of the $\mathfrak{D}$ modality considers only closed pairs within the time window to compute the average distance, so open pairs are to 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. Axioms A4, A5, A6, A7, A8 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 for evaluating a formula with the $\mathfrak{D}$ modality is shown in Fig. 5, which depicts a simple trace and the values assumed by the counters $c_{\phi,\psi}$, $h_{\phi,\psi}$, $s_{\phi,\psi}$, $a_{\phi,\psi}$, 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 of the sample trace, 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 with 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 with 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 with 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 with a hexagon in the figure); its value is $\frac{5}{1} = 5$.

3.5 Complexity of the translation

We conclude this section by briefly discussing the complexity of the proposed translation. The conversion of SOLOIST formulae into their single-sorted quantifier-free version over ω-words is linear in the cardinality of the sorts of the variables occurring in the initial formula. As for translation ρ, for the atomic propositions and all aggregate modalities except $\mathfrak{M}$ we introduce a fixed-length formula; notice that subformulae occuring in aggregate modalities are restricted to be atomic. The translation of the $\mathfrak{M}$ modality is linear in the parameter K. Overall, our translation is linear with respect to the size of the input formula.

4 Implementation

The translation described in the previous section has been implemented in a tool [25]; this tool acts as a front-end for translating SOLOIST formulae into the input format of the ZOT [29] verification toolset. ZOT supports satisfiability checking of CLTLB($\mathscr{D}$) formulae by means of SMT solvers; it performs the translation of its input language into a quantifier-free formula of the theory of uninterpreted functions with equality combined with integer difference logic (QF-EUF $\cup$ IDL). 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 use ZOT (together with the plugin we developed) to perform trace checking of service execution traces against requirements expressed in SOLOIST. Trace checking (also called *trace validation* [27] or *history checking* [19]) 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. In the context of this paper, a trace is composed by the events corresponding to the interactions of a composite service with its external services. We express this execution trace also as a temporal logic formula; at each time instant, an atomic proposition holds if and only if the corresponding service interaction event (e.g., the invocation of an operation on a partner service) occurred.

We now walk through the translation steps applied to a small 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. 6 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 Section 3.1; 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 $\phi \equiv \mathfrak{C}^{30}_{>5}(p)$ over the trace H of length 50 depicted in Fig. 6; trace checking is performed at time instant 30. The SOLOIST formula ϕ is translated into CLTLB($\mathscr{D}$) as $\rho(\phi) \equiv \mathsf{X}(c_p) - \mathsf{Y}^{29}(c_p) > 5$, 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 (respectively, an arithmetic proposition) to each sub-formula (respectively, arithmetical temporal term) in the input formula. For example, as shown in Fig. 6, the arithmetic propositions a and b correspond, respectively, to the arithmetical temporal terms $\mathsf{X}(c_p)$ and $\mathsf{Y}^{29}(c_p)$; c is an arithmetical proposition holding the value of the $\mathsf{X}(c_p) - \mathsf{Y}^{29}(c_p)$ arithmetic temporal term; proposition d corresponds to the entire input formula. The values of these auxiliary propositions are defined in each time instant $i = 0 \ldots 49$, 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 $\neg(\mathsf{X}^{30}(\rho(\phi))) \wedge \mathscr{C}_{c_p}$. Notice that the formula ϕ is negated; hence, it is satisfied by trace H if the SMT solver returns *unsat*; the exponent 30 in the term $\mathsf{X}^{30}(\rho(\phi))$ is determined by the evaluation of the formula fixed at time instant 30. We remark that details of the translation from CLTLB($\mathscr{D}$) to the input language of the SMT solver (as sketched in the bottom row of Fig. 6) have been omitted since they are out of the scope of this work; for them, we refer the reader to [29].

5 Evaluation

We evaluated the performance of our approach by executing different runs of trace checking of various SOLOIST formulae over synthesized traces. These traces were obtained using the Process Log Generator (PLG) tool [13] on a model of the running example from Sect. 2.2. This model was defined by specifying the workflow structure, the duration of each synchronous *invoke* activity, the branching probabilities, and the error rates. Other activities (e.g., *receive*) were given 0 as duration; branching was used to create loops and simulate the behavior of the *pick* activity. The PLG tool is able to synthesize logs of process invocations from its input model.

For each run of the trace checker, we recorded the memory usage, the translation time and the SMT verification time. In each run, we variated the following parameters:

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 *invoke* activities occurring in the trace as well as on the branching probabilities of the loop in the workflow.

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

Observation subinterval: it is used in the $\mathfrak{U}$ and $\mathfrak{M}$ modalities as the h parameter.

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 2 GB RAM allocated for the verification tool. We used

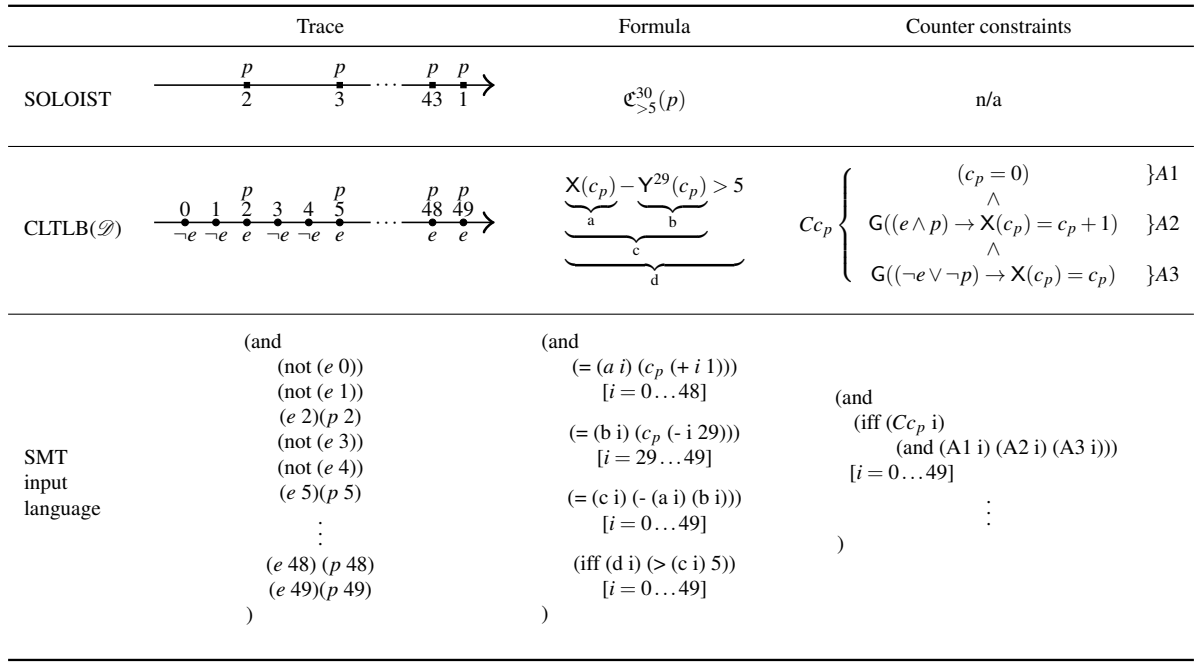


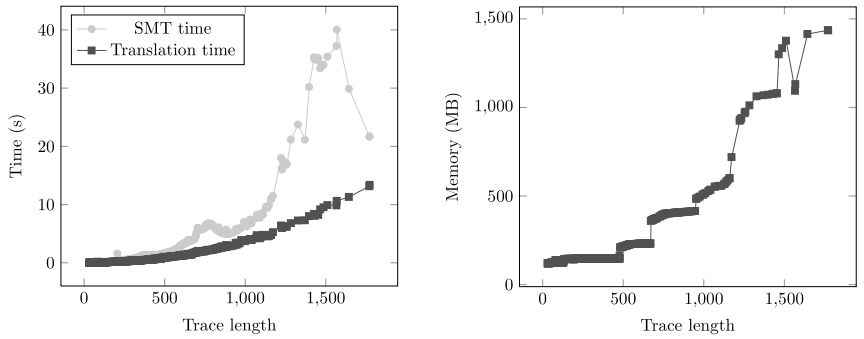
Figure 6: Example of the translation from SOLOIST to CLTLB($\mathscr{D}$) and then to the input language of the SMT solver

the *Z3* [16] SMT solver v. 4.3.1.

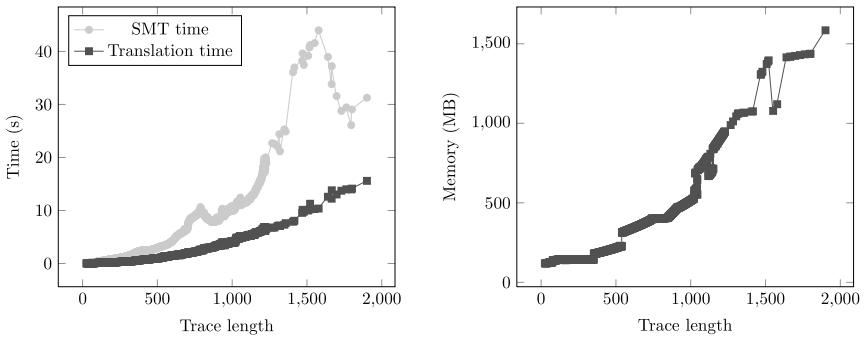
Scalability of the approach

In this section we describe the results of a set of runs, used to show how scalable our translation is with respect to the parameters mentioned above. 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.

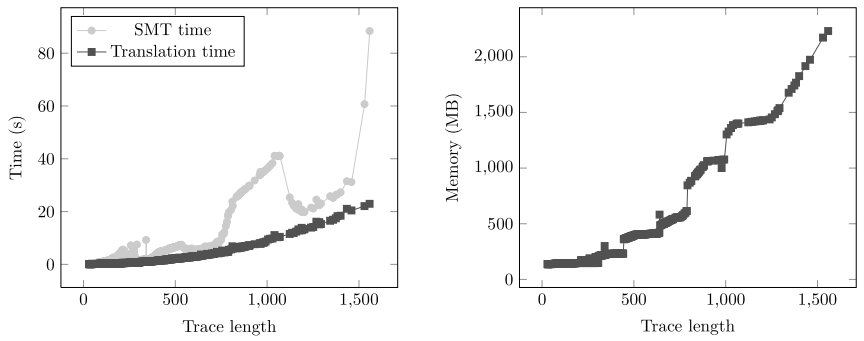
The first parameter we consider is the length H of the trace. We synthesized 19010 different traces, of variable length between 100 and 2000, corresponding to the process of the running example; we checked the following formulae on them: $\mathfrak{C}^{100}_{<30}(getBalance\_start)$, $\mathfrak{C}^{100}_{>30}(getBalance\_start)$, and $\mathfrak{D}^{100}_{>30}\{(getBalance\_start, getBalance\_end)\}$. The results of executing trace checking for each of these three formulae on the synthesized traces are shown in Fig. 7; in each column, the upper plot shows the time of the translation and the one taken by the SMT solver, while the lower plot shows the memory usage. 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 dedicated for the evaluation of formulae 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 formulae with the $\mathfrak{D}$ modality, the maximum number of time instances manageable before exhausting the preset memory was 2000. The lower value with respect to the case of 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. One can also notice that the growth of the resource usage is irregular; these irregularities are caused by the inner workings of the SMT solver.



Formula: $\mathfrak{C}^{100}_{<30}(p)$ (a)

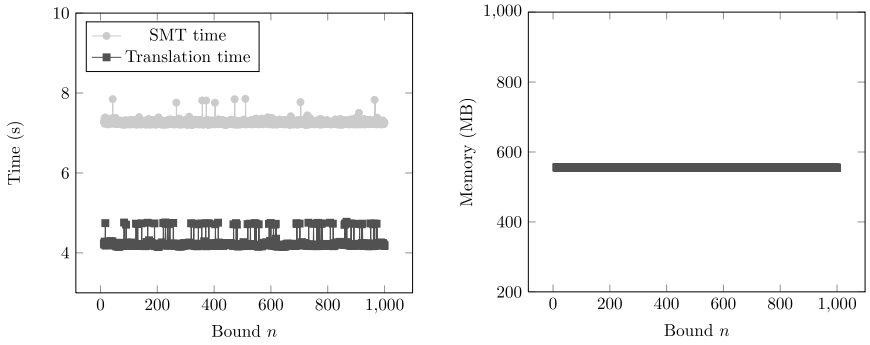


Formula: $\mathfrak{C}^{100}_{>30}(p)$ (b)

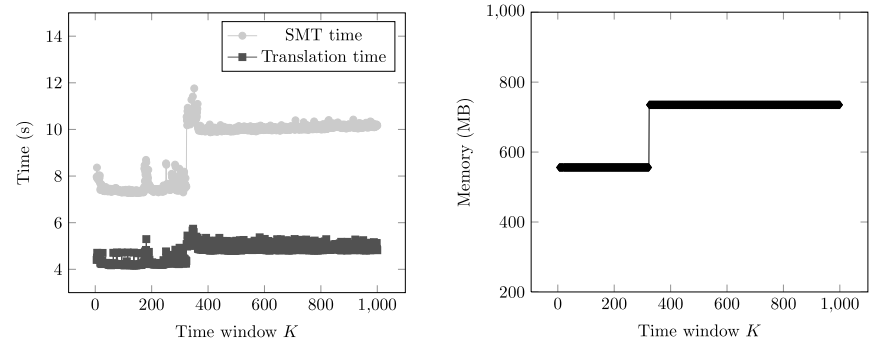


Formula: $\mathfrak{D}^{100}_{>30}\{(p,q)\}$ (c)

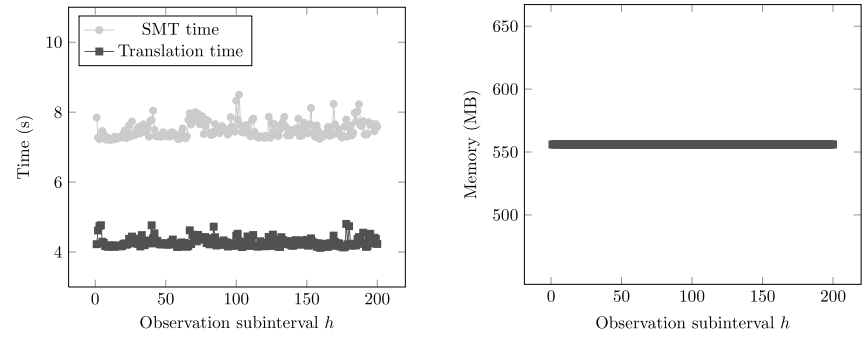
Figure 7: Scalability of the translation of the $\mathfrak{C}$ and $\mathfrak{D}$ modalities with respect to the length of the trace, where $p \equiv$ *getBalance_start* and $q \equiv$ *getBalance_end*



Parameter: Bound n Formula: $\mathfrak{C}^{1000}_{>n}(p)$ (a)



Parameter: Time window K Formula: $\mathfrak{C}^{K}_{>5}(p)$ (b)



Parameter: Subinterval h Formula: $\mathfrak{U}^{1000,h}_{<5}(q)$ (c)

Figure 8: Scalability of the translation with respect to the comparison operator bound n (a), length of the time window K (b), and the observation subinterval h (c), where $p \equiv$ *getBalance_start* and $q \equiv$ *deposit_start*

Table 1: Evaluation data of the running example

Property	ZOT time (s)	SMT time (s)	Total time (s)	Memory (MB)
	(mean/standard deviation)			
(QP1)	10.97/0.62	19.70/1.55	30.27/1.60	931.0/0
(QP2)	20.77/1.37	107.30/2.69	125.50/2.29	1261.7/118.24
(QP3)	10.56/0.46	31.89/1.74	42.08/1.73	793.0/0
(¬ QP1)	11.59/0.75	28.00/1.67	38.94/1.73	932.0/0
(¬ QP2)	20.74/0.69	324.80/159.1	343.00/158.2	1210.8/80.47
(¬ QP3)	11.10/0.53	31.37/0.55	41.97/0.70	942.0/0

In our evaluation, we noticed that the other parameters, namely the length of the time window K, the observation subinterval h, and the bound of the comparison operator n, do not affect the resources usage, and only introduce some non-deterministic noise in the SMT solver time. This can be seen in Fig. 8, which shows the time and memory usage with respect to the variation of each of these parameters when checking formulae over a synthesized trace of length fixed to 1000; in (a) the formula checked is $\mathfrak{C}^{1000}_{>n}(\mathit{getBalance\_start})$, in (b) it is $\mathfrak{C}^{K}_{>5}(\mathit{getBalance\_start})$, and in (c) it is $\mathfrak{U}^{1000,h}_{<5}(\mathit{deposit\_start})$.

Application to the running example

Here we report on the use of trace checking to assess whether the SOLOIST properties defined in Sect. 2.2 hold for the executions of the business process described in the running example. We performed the checks on 10 traces of length 1000 that we picked randomly from the set of synthesized traces. We checked the three properties as well as their negations; the average time and memory usage, as well as their standard deviation, are reported in table 1; these results show the practical feasibility of our approach.

Limitations

As a result of the evaluation of our approach, we identified a limitation in its application in the context of trace checking. While the complexity of the translation itself depends on the input formula, the performance of trace checking, as shown above, depends on the length of the trace processed by the SMT solver. Notice that this trace is the one resulting from the conversion of the input trace (defined as a timed ω-word) into the CLTLB($\mathscr{D}$) equivalent (defined as an ω-word). When the input trace is sparse, the conversion expands it into a very long ω-word, populated with many not-valid time instants (i.e., where $\neg e$ holds). This limitation will be addressed in future work, by assessing other target languages different from CLTLB($\mathscr{D}$).

6 Related Work

In this section we briefly survey existing work on trace checking and verification of quantitative properties specified in languages similar to SOLOIST. For a discussion on SOLOIST and related specification languages see [11].

Finkbeiner et al. [20] 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. However, these automata are manually built from the specification; this makes applying frequent changes to the property error-prone.

In [6] 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 occurences of relations in the temporal first-order structure.

The trace checking approach presented in [5] 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 [7], 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 SOLOIST $\mathfrak{C}$ modality is that there is no timing information: this means one cannot specify the exact part of the trace the modality should consider.

In [15], 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 [15] 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.

7 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 [10]. The SOLOIST language was developed [11] to express these patterns, which involve aggregate operations on events occurring in a given time window. These operators make SOLOIST a quite expressive domain-specific specification language. Expressivity, however, often complicates verification and sometimes makes it practically intractable or even impossible to achieve mechanically. In our previous work, automatic verification was proved to be conceptually possible. In this work, we showed that it can be done effectively. We achieved that by an efficient translation of SOLOIST into CLTLB($\mathscr{D}$), a variant of linear temporal logic that supports counter variables. The formulae resulting from this translation have a succinct encoding, which can be efficiently exploited to perform trace checking of service execution traces against quantitative temporal properties expressed in SOLOIST. We detailed the results of evaluating the achieved scalability, with respect to several parameters.

The use of SOLOIST in the context of realistic and practical verification activities is the goal of further on-going research. In particular, we are currently exploring an alternative translation based on other logics, such as CLTL-o.c. (Constrained LTL over clocks) [9], which is currently under scrutiny

within our research group. This new encoding would support variable distance between discrete events; this feature could then be used to encode more succinctly long and sparse traces, which we found to be rather common in practice. 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 Web service monitoring framework such as Dynamo [21].

Acknowledgments

This work has been partially supported by the European Community under the IDEAS-ERC grant agreement no. 227977-SMScom and by the National Research Fund, Luxembourg (FNR/P10/03). The authors wish to thank Marcello Bersani and Matteo Pradella for their precious help with ZOT.

References

[1] R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. Comput.*, 104(1):35–77, 1993.

[2] T. Andrews et al. Business Process Execution Language for Web Services, Version 1.1, 2003.

[3] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. Validation of web service compositions. *IET Softw.*, 1(6):219–232, 2007.

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

[5] 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*, volume 7687 of *LNCS*, pages 184–198. Springer, 2013.

[6] D. Basin, F. Klaedtke, S. Marinovic, and E. Zălinescu. Monitoring of temporal first-order properties with aggregations. In *Proc. of RV'13*, LNCS. Springer, 2013. to appear.

[7] A. Bauer, R. Goré, and A. Tiu. A first-order policy language for history-based transaction monitoring. In *Proc. of ICTAC '09*, volume 5684 of *LNCS*, pages 96–111. Springer, 2009.

[8] M. M. Bersani, A. Frigeri, A. Morzenti, M. Pradella, M. Rossi, and P. San Pietro. Constraint LTL Satisfiability Checking without Automata. *CoRR*, abs/1205.0946, 2012.

[9] M. M. Bersani, M. Rossi, and P. San Pietro. A tool for deciding the satisfiability of continuous-time metric temporal logic. In *Proc. of TIME 2013*, 2013. to appear.

[10] D. Bianculli, C. Ghezzi, C. Pautasso, and P. Senti. Specification patterns from research to industry: a case study in service-based applications. In *Proc. of ICSE 2012*, pages 968–976. IEEE Computer Society, 2012.

[11] D. Bianculli, C. Ghezzi, and P. San Pietro. The tale of SOLOIST: a specification language for service compositions interactions. In *Proc. of FACS'12*, volume 7684 of *LNCS*, pages 55–72. Springer, 2013.

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

[13] A. Burattin and A. Sperduti. Plg: A framework for the generation of business process models and their execution logs. In *Business Process Management Workshops*, volume 66 of *LNBIP*, pages 214–219. Springer, 2011.

[14] G. Canfora and M. Di Penta. Service oriented architectures testing: a survey. In *ISSSE 2006–2008*, volume 5413 of *LNCS*, pages 78–105. Springer, 2009.

[15] L. de Alfaro. Temporal logics for the specification of performance and reliability. In *Proc. of STACS'97*, volume 1200 of *LNCS*, pages 165–176. Springer, 1997.

[16] L. M. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Proc. of TACAS 2008*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[17] S. Demri and D. D'Souza. An automata-theoretic approach to constraint LTL. *Inf. Comput.*, 205(3):380–415, 2007.

[18] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property specification patterns for finite-state verification. In *Proc. of FMSP '98*, pages 7–15. ACM, 1998.

[19] M. Felder and A. Morzenti. Validating real-time systems by history-checking TRIO specifications. *ACM Trans. Softw. Eng. Methodol.*, 3(4):308–339, Oct. 1994.

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

[21] C. Ghezzi and S. Guinea. Run-time monitoring in service-oriented architectures. In Baresi and Di Nitto [4], pages 237–264.

[22] S. Hallé, R. Villemaire, and O. Cherkaoui. Specifying and validating data-aware temporal web service properties. *IEEE Trans. Softw. Eng.*, 35(5):669–683, 2009.

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

[24] S. Konrad and B. H. C. Cheng. Real-time specification patterns. In *Proc. of ICSE '05*, pages 372–381. ACM, 2005.

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

[26] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Proc. of Logics of Programs*, volume 193 of *LNCS*, pages 196–218. Springer, 1985.

[27] A. Mrad, S. Ahmed, S. Hallé, and E. Beaudet. Babeltrace: A collection of transducers for trace validation. In *Proc. of RV 2012*, volume 7687 of *LNCS*, pages 126–130. Springer, 2013.

[28] 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*, pages 312–320. ACM, 2007.

[29] M. Pradella, A. Morzenti, and P. San Pietro. Bounded satisfiability checking of metric temporal logic specifications. *ACM Trans. Softw. Eng. Methodol.*, 22(3):20:1–20:54, July 2013.

[30] G. Salaün. Analysis and verification of service interaction protocols - a brief survey. In *Proc. of TAV-WEB 2010*, volume 35 of *EPTCS*, pages 75–86, 2010.