A Dynamic Temporal Logic for Quality of Service in Choreographic Models *

Carlos G. Lopez Pombo**[1,2], Agustín E. Martinez Suñé[3], Emilio Tuosto[4]

[1] Centro Interdisciplinario de Telecomunicaciones, Electrónica, Computación y Ciencia Aplicada - CITECCA, Universidad Nacional de Río Negro - Sede Andina
[2] CONICET–UBA. Instituto de Investigación en Ciencias de la Computación
[3] Universidad de Buenos Aires, FCEyN, Departmento de computación
[4] Gran Sasso Science Institute

Abstract We propose a framework for expressing and analyzing the *Quality of Service* (QoS) of message-passing systems using a choreographic model that consists of *g-choreographies* and *Communicating Finite State machines* (CFSMs). The following are our three main contributions: (I) an extension of CFSMs with non-functional contracts to specify quantitative constraints of local computations, (II) a dynamic temporal logic capable of expressing QoS, properties of systems relative to the g-choreography that specifies the communication protocol, (III) the semi-decidability of our logic which enables a bounded model-checking approach to verify QoS property of communicating systems.

1 Introduction

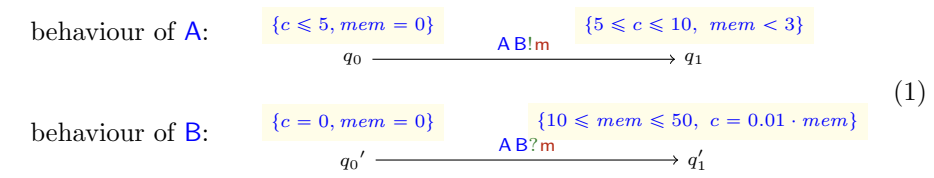
Over the past two decades, software has steadily changed from monolithic applications to distributed cooperating components. Choreographic approaches are gaining momentum in industry (e.g. [1,2,3,5]) which, increasingly, conceives applications as components interacting over existing communication infrastructures. Among other models, choreographies stand out for a neat separation of concerns: choreographic models abstract away local computations from communications among participants. In fact, since their introduction [6], choreographies

* Research partly supported by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement No 778233. Research partly supported by the PRO3 MUR project Software Quality, and PNRR MUR project VITALITY (ECS00000041), Spoke 2 ASTRA - Advanced Space Technologies and Research Alliance. Carlos G. Lopez Pombo's research is partly supported by Universidad de Buenos Aires by grant UBACyT 20020170100544BA and Agencia Nacional de Promoción de la Investigación, el Desarrollo Tecnológico y la Innovación Científica through grant PICT-2019-2019-01793.

The authors thank the anonymous reviewers for their constructive comments.

** On leave from Instituto de Ciencias de la computación CONICET–UBA and Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires.

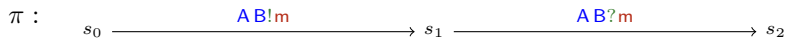
advocate for a separation between a *global view* and a *local view* of communication. The former is a high-level description of (distributed) interactions. The latter view is a description of each component in isolation. This is the distinctive feature of choreographies that we exploit here to reason about quantitative properties of applications. The basic idea is to specify the values of quality attributes of local states of components and then *aggregate* those attributes along runs involving communications. A simple example can illustrate this. Suppose that a component A sends another component B a message m and we want to consider two quality attributes: monetary cost (c) and memory consumption (mem). This behaviour can be abstracted away with the finite-state machines below



(1)

where A B!m and A B?m respectively denote the output and input communication actions, and each state is decorated with a specification predicating over the quality attributes in the local states of A and B. For instance, both A and B allocate no memory in their initial states, computation in A may cost up to five monetary units before executing the output, and B has no cost since it's just waiting to execute the input ($c = 0$). Likewise, after the communication actions, the local computations of A and B are specified by the formulae associated to states q_1 and q_1'.

The interaction between A and B depends on the communication infrastructure; e.g., asynchronous message-passing yields a run like



where first the message is sent by A and then it is eventually received by B.

We are interested in analyzing quality properties that admit a measurement, thus assuming that the QoS attributes are *quantitative*. These properties encompass both quantitative attributes at the application level as well as resource consumption metrics. For instance, we could be interested in analyzing the monetary cost or the number of messages retrieved in a messaging system; but we could also be interested in analyzing its memory usage or CPU time. It's important to emphasize that our framework is designed to be agnostic and adaptable, allowing for the consideration of any quantifiable attribute, regardless of its specific nature. Furthermore, our framework is specifically designed to enable analysis of how quantitative properties of local computations influence the system-wide properties. Hence, we envisage the quality constraints as *contracts* that the local computations of components should honour. For instance, the specifications in (1) tell the cost of local computations in A and B, they do not predicate on the QoS of the communication infrastructure.

Once these quality constraints on local computations are fixed, natural questions to ask are e.g., "is the memory consumption of B along run π within a given

range?" or "is the monetary overall monetary cost below a given threshold?". Answers to such questions require checking that the *aggregation* of the constraints on the quality attributes along the run π entails the properties. Interestingly, how to aggregate those constraints depends on the quality attributes. For instance, the aggregation of memory consumption can be computed by taking the maximum, while the aggregation of monetary cost can be computed as the sum. We work under the hypothesis that developers have no control over communication infrastructure. More precisely, QoS aspects related to how communications are realised are not under the control of applications' designers. Instead, designers have control over local computations, thus suggesting that QoS constraints are naturally associated to states of components. Indeed, we rely on behavioural types (such as [9,10,11,12,13]) which abstract away low level details.

Contributions We propose a framework for the design and analysis of QoS-aware distributed systems, enabled by the following technical contributions:

Models for QoS attributes. Section 4 presents a straightforward extension of communicating finite-state machines (CFSMs [14]; reviewed in Section 3) to express QoS aspects of components. Basically, we assign to each state of CFSMs a QoS specification as in (1).
We adopt real-closed fields (RCFs, cf. Section 3) to abstractly represent QoS values; besides being a complete and decidable abstract formalisation of the first-order theory of the real numbers, RCFs are instrumental for a smooth definition of our framework.

A dynamic temporal logic for QoS. Section 5 introduces a logic, dubbed $\mathcal{QL}$, to express QoS properties. Taking inspiration from *Propositional Dynamic Linear Temporal Logic* (DLTL) [15], $\mathcal{QL}$ indexes temporal modalities with *global choreographies* [16] (g-choreographies, Section 3), a model of global views of choregraphies, in order to predicate over QoS properties of the whole system. This is a distinct characteristic of $\mathcal{QL}$ that we comment in Section 2.

A semi-decision procedure for $\mathcal{QL}$. Section 6 proves $\mathcal{QL}$ to be semi-decidable by providing a k-bounded semi-decision procedure and relying on the decidability of the theory of real-closed fields [17] to check QoS constraints in atomic formulae. A distinctive aspect of the procedure is that it can be used as a bounded model-checking procedure where the bound is established by the g-choreographies indexing the modalities of $\mathcal{QL}$ formulae.

Section 7 draws some conclusions and points out some further lines of research.

2 Related Work

The relevance of the problem addressed here has been already highlighted by other researchers [7,8]. There is a vast literature on QoS, spanning a wide range of contexts and methods [18,19]. This paper can be positioned in the category of general application-level QoS. The combination of RCFs and our behavioural

types aims to capture essential aspects for applications' quantitative analysis while striving for generality. In this vein, a proof-of-concept methodology based on behavioural types has been proposed in [20] for client-server systems. To the best of our knowledge, our is the first work blending behavioural types with QoS and offering a decision procedure for multiparty protocols.

In order to abstractly capture QoS (instead of focusing on specific attributes) we adopt RCFs. Other abstract models of QoS such as quantales [21] or c-semirings [22,23,24] have been proposed. We opted for RCFs due to their inherent decidability, which is crucial for ensuring the decidability of our logic. Moreover, RCFs offer practical advantages as they can be readily employed in modern SMT (satisfiability modulo theories) solvers [41, Chapter 33].

Theory presentations over QoS attributes are used in [25] to enable the automatic analysis of QoS properties with a specification language that only considers convex polytopes; this restriction is not present in our language. Also, the approach in [25] can be thought as "monolithic", in the sense that specifications are given considering the system as a black box. We instead assign QoS contracts to states of components and then aggregate them in order to analyze properties along executions of the behavior emerging from interactions.

The use of choreographic methods for non-functional analysis yields other advantages. For instance, QoS contracts of components are derived from global specifications [7]. These contracts can then be used for run-time prediction, adaptive composition, or compliance checking, similarly to what is done in [8]. This top-down approach can be transferred to behavioural types as well similarly to what has been done in [11,26] for qualitative properties. The framework proposed in [27] uses CFSMs as a dynamic binding mechanism of services but only considers the communicational aspects of the software component. Such a framework could be extended to include QoS attributes as well by leveraging the results presented in this paper.

Our $\mathcal{QL}$ logic takes inspiration from *dynamic linear temporal logic* (*DLTL*) [15] which blends trace semantics (akin *linear temporal logic* [28]) and regular expressions over a set of atomic actions (akin *propositional dynamic logic* [29]). Intuitively a key difference is that, unlike *DLTL*, $\mathcal{QL}$ does not predicate about the behaviour of sequential programs; rather $\mathcal{QL}$ describes properties of asynchronous message-passing systems. This requires a modification of the syntax of *DLTL*; in fact, the syntax of $\mathcal{QL}$ is essentially the same of *DLTL* barred for the indexes of modalities, which become choreographies of interactions. This straightforward modification has deep impact on the semantics which requires a complete redefinition (see Section 7 for further details). Another key difference is that, while *DLTL* is propositional, $\mathcal{QL}$'s atomic formulae are first order formulae on QoS attributes. As a consequence, not only $\mathcal{QL}$ can express usual temporal properties, such as safety and liveness ones, but temporal properties constraining the value of QoS attributes. These points of comparison with *DLTL* apply in the same way to a similar logic called *linear dynamic logic* (LDL), introduced first in [30] and later formalized for finite traces in [31].

3 Preliminaries

This section surveys background material underpinning our work. We first describe the protocol used as a running example, then we review our choreographic model and we briefly recall *real-closed fields*.

A running example. Through the paper we will use a (simplified variant) of the POP protocol [32]. This protocol allows mail clients to access a remote mailbox and retrieve e-mails. In the POP protocol a client starts the communication by sending a message of type helo to a POP server (note that protocol specifications are oblivious of messages' payload).[5] The server replies with the number of unread messages in the mailbox using a message of type int. At this point the client can either halt the protocol or read one of the e-mails. These options are selected by sending a message of type quit or of type read respectively. In the former case, the server acknowledges with a message of type bye and the protocol ends. In the latter case, the server sends the client the number of bytes of the current unread message in a message of type size. Next, the client has again a choice between quitting the protocol (as before) or receiving the email (selected in the read message) by sending the server a message of type retr. In the latter case the server sends the email with a message of type msg, the client answers with a message of type ack and the reading process starts again.

A choreographic model. We use *global choreographies* [16] to specify the global view of communicating systems whose local view are rendered as *communicating finite state machines* [14].

Hereafter, we fix a set $\mathcal{P}$ of *participants* and a set $\mathcal{M}$ of (types of) *messages* such that $\mathcal{P} \cap \mathcal{M} = \varnothing$. We start by surveying the definition of g-choreographies.

Definition 1 (Global choreographies [16]). *A* global choreography over $\mathcal{P}$ and $\mathcal{M}$ *(*g-choreography *for short) is a term* G *that can be derived in*

$$\mathsf{G} \;::=\; \mathbf{0} \;\mid\; \mathsf{A}{\to}\mathsf{B}\colon \mathsf{m} \;\mid\; \mathsf{G};\mathsf{G}' \;\mid\; \mathsf{G} \mid \mathsf{G}' \;\mid\; \mathsf{G} + \mathsf{G}' \;\mid\; \mathsf{G}^*$$

where $\mathsf{A}, \mathsf{B} \in \mathcal{P}$, $\mathsf{A} \neq \mathsf{B}$ *and* $\mathsf{m} \in \mathcal{M}$.

Intuitively, a g-choreography specifies the communication protocol of participants. The basic g-choreography is the empty one $\mathbf{0}$, which specifies that no communications should happen. An *interaction* $\mathsf{A}{\to}\mathsf{B}\colon \mathsf{m}$ specifies that participants A and B (are expected to) exchange a message of type m; it is worth remarking that we assume asynchronous communication where the sender A does not wait for B to consume m to continue its execution. Moreover, g-choreographies can be composed sequentially ($\mathsf{G};\mathsf{G}'$), in parallel ($\mathsf{G} \mid \mathsf{G}'$), and in non-deterministic choices ($\mathsf{G} + \mathsf{G}'$); we assume that $\mathbf{0}$ is the neutral element of $\_;\_$, $\_ \mid \_$, and $\_ + \_$. Note that, due to asynchrony in the communication, in a sequential composition $\mathsf{G};\mathsf{G}'$, outputs in G' can occur before G is fully executed; for instance, the distributed execution of $\mathsf{A}{\to}\mathsf{B}\colon \mathsf{m};\mathsf{C}{\to}\mathsf{B}\colon \mathsf{m}'$ allows the output from C to happen before the one from A. Finally, a g-choreography may be iterated (G^*).

[5] Our framework can handle multiparty protocols; however, our examples are two-party for simplicity. Also, we stick to the types of messages as carefully described in the protocol specifications [32].

Example 1 (A g-choreography for POP). Our running example can be expressed as the g-choreography $\mathsf{G}_{\text{pop}} = \mathsf{C}{\to}\mathsf{S}\colon \mathsf{helo}; \mathsf{G}_{\text{start}} + \mathsf{G}_{\text{quit}}$ where

$$\mathsf{G}_{\text{start}} = \mathsf{S}{\to}\mathsf{C}\colon \mathsf{int}; (\mathsf{G}_{\text{read}} + \mathsf{G}_{\text{read}}; \mathsf{G}_{\text{retr}})^*; \mathsf{G}_{\text{quit}} \qquad \mathsf{G}_{\text{read}} = \mathsf{C}{\to}\mathsf{S}\colon \mathsf{read}; \mathsf{S}{\to}\mathsf{C}\colon \mathsf{size}$$
$$\mathsf{G}_{\text{retr}} = \mathsf{C}{\to}\mathsf{S}\colon \mathsf{retr}; \mathsf{S}{\to}\mathsf{C}\colon \mathsf{msg}; \mathsf{C}{\to}\mathsf{S}\colon \mathsf{ack} \qquad \mathsf{G}_{\text{quit}} = \mathsf{C}{\to}\mathsf{S}\colon \mathsf{quit}; \mathsf{S}{\to}\mathsf{C}\colon \mathsf{bye}$$

($\_;\_$ takes precedence over $\_ + \_$). ◇

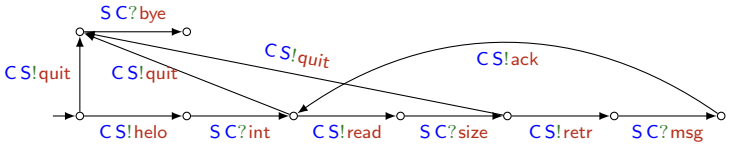
The participants of a communicating system interact through *channels* borrowed from the set $\mathcal{C} = \{ (\mathsf{A}, \mathsf{B}) \in \mathcal{P} \times \mathcal{P} \mid \mathsf{A} \neq \mathsf{B} \}$. A channel $(\mathsf{A}, \mathsf{B}) \in \mathcal{C}$ (written $\mathsf{A}\,\mathsf{B}$ for short) allows A to asynchronously send messages to B through an unbounded FIFO buffer associated to $\mathsf{A}\,\mathsf{B}$. The set of *communication actions* is $\mathcal{L} = \mathcal{L}^! \cup \mathcal{L}^?$ where $\mathcal{L}^! = \{\mathsf{A}\,\mathsf{B}!\mathsf{m} \mid \mathsf{A}\,\mathsf{B} \in \mathcal{C} \text{ and } \mathsf{m} \in \mathcal{M}\}$ and $\mathcal{L}^? = \{\mathsf{A}\,\mathsf{B}?\mathsf{m} \mid \mathsf{A}\,\mathsf{B} \in \mathcal{C} \text{ and } \mathsf{m} \in \mathcal{M}\}$ are respectively the set of *output* and *input* actions. The *language* $\mathcal{L}[\![\mathsf{G}]\!]$ of a g-choreography G is essentially the set of all possible sequences in $\mathcal{L}$ compatible with the causal relation induced by G. Since $\mathcal{L}[\![\mathsf{G}]\!]$ is prefix-closed, we write $\hat{\mathcal{L}}[\![\mathsf{G}]\!]$ for the set of sequences in $\mathcal{L}[\![\mathsf{G}]\!]$ that are not proper prefixes of any other sequence in $\mathcal{L}[\![\mathsf{G}]\!]$. The technical definition of $\mathcal{L}[\![\mathsf{G}]\!]$, immaterial here, can be found in [16]. We will adapt CFSM [14] to model the QoS-aware *local view* of a system.

Definition 2 (Communicating systems [14]). *A* communicating finite-state machine *(*CFSM*) is a finite transition system* $M = (Q, q_0, \to)$ *where*

- Q *is a finite set of* states *with* $q_0 \in Q$ *the* initial *state, and*
- $\to \subseteq Q \times \mathcal{L} \times Q$*; we write* $q \xrightarrow{\ell} q'$ *for* $(q, \ell, q') \in \to$.

For $\mathsf{A}\,\mathsf{B}!\mathsf{m} \in \mathcal{L}$ *(resp.* $\mathsf{A}\,\mathsf{B}?\mathsf{m} \in \mathcal{L}$*), let* $\mathit{sbj}(\mathsf{A}\,\mathsf{B}!\mathsf{m}) = \mathsf{A}$ *(resp.* $\mathit{sbj}(\mathsf{A}\,\mathsf{B}?\mathsf{m}) = \mathsf{B}$*). Given* $\mathsf{A} \in \mathcal{P}$*,* M *is* A-local *if* $\mathit{sbj}(\ell) = \mathsf{A}$ *for each* $q \xrightarrow{\ell} q'$*.* A (communicating) system *is a map* $S = (M_{\mathsf{A}})_{\mathsf{A} \in \mathcal{P}}$ *assigning a* A*-local CFSM* M_{A} *to each* $\mathsf{A} \in \mathcal{P}$.

Example 2 (Communicating system for POP). The following CFSM exhibits a behaviour of a POP client compatible with the protocol in Example 1 because its executions yield a subset of the client's execution specified there.



For a POP server, being a two-party protocol, we can use a dual CFSM obtained by replacing send actions with corresponding receive actions and vice versa. ◇

The asynchronous communication between participants is formalised by a labelled transition system (LTS) tracking the (local) state of each CFSM and the content of each buffer (i.e. communication channel) in the system. A *configuration* of a communicating system S is a pair $s = \langle \mathfrak{q} ; \mathfrak{b} \rangle$ where $\mathfrak{q}$ and $\mathfrak{b}$ respectively map participants to states and channels to sequences of messages;

state $\mathfrak{q}(\mathsf{A})$ keeps track of the state of the machine M_{A} and buffer $\mathfrak{b}(\mathsf{A}\,\mathsf{B})$ yields the messages sent from A to B and not yet consumed. The *initial* configuration s_0 is the one where, for all $\mathsf{A} \in \mathcal{P}$, $\mathfrak{q}(\mathsf{A})$ is the initial state of the corresponding CFSM and $\mathfrak{b}(\mathsf{A}\,\mathsf{B})$ is the empty sequence for all $\mathsf{A}\,\mathsf{B} \in \mathcal{C}$.

A configuration $s' = \langle \mathfrak{q}' \,;\, \mathfrak{b}' \rangle$ is *reachable* from another configuration $s = \langle \mathfrak{q} \,;\, \mathfrak{b} \rangle$ by *firing a transition* ℓ, written $s \overset{\ell}{\Longrightarrow} s'$, if there is a message $\mathsf{m} \in \mathcal{M}$ such that either (1) or (2) below holds:

1. $\ell = \mathsf{A}\,\mathsf{B}!\mathsf{m}$ with $\mathfrak{q}(\mathsf{A}) \overset{\ell}{\rightarrow}_{\mathsf{A}} q'$ and
   a. $\mathfrak{q}' = \mathfrak{q}[\mathsf{A} \mapsto q']$
   b. $\mathfrak{b}' = \mathfrak{b}[\mathsf{A}\,\mathsf{B} \mapsto \mathfrak{b}(\mathsf{A}\,\mathsf{B}).\mathsf{m}]$
2. $\ell = \mathsf{A}\,\mathsf{B}?\mathsf{m}$ with $\mathfrak{q}(\mathsf{B}) \overset{\ell}{\rightarrow}_{\mathsf{B}} q'$ and
   a. $\mathfrak{q}' = \mathfrak{q}[\mathsf{B} \mapsto q']$ and
   b. $\mathfrak{b} = \mathfrak{b}'[\mathsf{A}\,\mathsf{B} \mapsto \mathsf{m}.\mathfrak{b}'(\mathsf{A}\,\mathsf{B})]$.

Condition (1) puts m on channel AB, while (2) gets m from channel AB. In both cases, any machine or buffer not involved in the transition is left unchanged in the new configuration s'.

Example 3 (Semantics of CFSMs). For the run π of the communicating system in (1) (cf. Section 1) we have, for $i \in \{0, 1, 2\}$, $s_i = \langle \mathfrak{q}_i \,;\, \mathfrak{b}_i \rangle$, where $\mathfrak{q}_0 = \{\mathsf{A} \mapsto q_0, \mathsf{B} \mapsto q_0'\}$, $\mathfrak{b}_0 = \{\mathsf{A}\,\mathsf{B} \mapsto \epsilon, \mathsf{B}\,\mathsf{A} \mapsto \epsilon\}$, $\mathfrak{q}_1 = \{\mathsf{A} \mapsto q_1, \mathsf{B} \mapsto q_0'\}$, $\mathfrak{b}_1 = \{\mathsf{A}\,\mathsf{B} \mapsto \mathsf{m}, \mathsf{B}\,\mathsf{A} \mapsto \epsilon\}$, and $\mathfrak{q}_2 = \{\mathsf{A} \mapsto q_1, \mathsf{B} \mapsto q_1'\}$, $\mathfrak{b}_2 = \mathfrak{b}_0$. ◇

Let S be a communicating system. A sequence $\pi = (s_i, \ell_i, s_{i+1})_{i \in I}$ where I is an initial segment of natural numbers (i.e., $i - 1 \in I$ for each $0 < i \in I$) is a run of S if $s_i \overset{\ell_i}{\Longrightarrow} s_{i+1}$ is a transition of S for all $i \in I$. The set of runs of S is denoted as Δ_S^∞ and the set of runs of length k is denoted as Δ_S^k. Note that Δ_S^∞ may contain runs of infinite length, the set of finite runs of S is the union of all Δ_S^k and will be denoted as Δ_S. Given a run π, we define $\mathcal{L}[\pi]$ to be the sequence of labels $(\ell_i)_{i \in I}$. The *language* of S is the set $\mathcal{L}[S] = \{\mathcal{L}[\pi] \mid \pi \in \Delta_S^\infty\}$. Finally, $prf : \Delta_S^\infty \to 2^{\Delta_S}$ maps each run $\pi \in \Delta_S^\infty$ to its set of finite prefixes. As usual, for all $\pi \in \Delta_S^\infty$, the empty prefix ϵ belongs to $prf(\pi)$. For convenience, we will occasionally write $s_0 \overset{\ell_0}{\Longrightarrow} s_1 \;\ldots\; s_n \overset{\ell_n}{\Longrightarrow} s_{n+1}$ for finite sequences.

Real-closed fields. Real numbers are natural candidates to express quantitative attributes of a software artifact. We adopt *real-closed fields* (RCFs), which is the formalisation of the first-order theory of the real numbers, as a foundation for QoS values. Let Σ_{field} denote the first-order signature $\langle \{0, 1\}, \{+, \cdot\}, \{<\} \rangle$. An ordered field is a first-order theory presentation $\langle \Sigma_{\text{field}}, \Gamma_{\text{field}} \rangle$, where Γ_{field} consists of the *field* axioms as well as the axioms defining $<$ as a strict total order relation. Real-closed fields are ordered fields whose non-empty subsets all have a supremum. Tarski's axiomatization of real-closed fields, denoted here as $\langle \Sigma_{\text{RCF}}, \Gamma_{\text{RCF}} \rangle$, was introduced in [17]. Tarski further demonstrated the existence of a decision procedure for this first-order theory of real numbers in [17, Thm. 37]. Thus, the main reason for selecting RCFs as the foundation for QoS lies in the fact that first-order theories extending them using elementary operations are decidable, providing effective means for analysis.

4 Quality of Service of Communicating Systems

In this section we extend CFSMs with QoS specifications in order to express QoS contracts of components in message-passing systems. Basically, each state of CFSMs is assigned a QoS contract specifying the usage of computational resources. We formalise QoS contracts as *QoS specifications* which are theory presentations over the RCFs, noted as $\langle \Sigma, \Gamma \rangle$, paired up with *aggregation operators*, noted as $\oplus^{\mathsf{a}}$, to define how each QoS attribute accumulates along a communicating system. These aggregation operators will be essential to formally define the notion of aggregation along a run, as shown later in Example 9.

Definition 3. *A* QoS specification $\langle \Sigma, \Gamma \rangle$ *is a (first-order) theory presentation extending* $\langle \Sigma_{\text{RCF}}, \Gamma_{\text{RCF}} \rangle$ *as follows:*

1. $\Sigma = \langle \{0,1\} \cup \mathsf{Q}, \{+,\cdot\} \cup \{\oplus^{\mathsf{a}}\}_{\mathsf{a} \in \mathsf{Q}}, \{<\} \rangle$*, where* Q *is a finite set of constant symbols (other that* 0 *and* 1*) representing the* quantitative attributes *(from now on referred to as* QoS attributes*) and, for each* $\mathsf{a} \in \mathsf{Q}$*,* $\oplus^{\mathsf{a}}$ *is an associative algebraic binary operator and*
2. $\Gamma = \Gamma_{\text{RCF}} \cup \Gamma'$*, being* Γ' *a finite set of first-order formulae formalising specific constraints over the QoS attributes in* Q*.*

The class of QoS specifications will be denoted as $\mathcal{C}(\mathsf{Q})$*.*

In order to preserve decidability of QoS properties, we only consider QoS specifications involving additional constant symbols representing the QoS attributes of components. Aggregation operators are required to be algebraic because the extension of the theory must be kept in the first-order fragment (uninterpreted function or predicate symbols must be avoided to preserve decidability). It is worth noticing that aggregation operators strongly depend on the nature of each specific attribute; for example, natural aggregation operators for memory and time are the maximum function and sum respectively.

Example 4 (QoS specification). With reference to Example 2, possible quantitative attributes of interest in an implementation of POP are $\mathsf{Q} = \{t, c, m\}$ representing CPU *t*ime, monetary *c*ost, and *m*emory usage, respectively. Then a QoS specification that characterises low computational costs, where no internal process consumes significant amount of resources, can be written according to Definition 3 as follows:

$$\Sigma = \langle \{0,1\} \cup \{t,c,m\}, \{+,\cdot\} \cup \{\oplus^t, \oplus^c, \oplus^m\}, \{<\} \rangle$$
$$\Gamma = \Gamma_{\text{RCF}} \cup \Gamma'_{\text{Low}}$$

where $\oplus^t = +$, $\oplus^c = +$, $\oplus^m = max$, and $\Gamma'_{\text{Low}} = \{t \leqslant .01, c \leqslant .01, m \leqslant .01\}$ ⋄

From now on, we fix a set of constant symbols Q which we omit from the set of *QoS specifications*, that will be referred to just as $\mathcal{C}$. It is worth noting that, when Q is fixed, a QoS specification $\langle \Sigma, \Gamma_{\text{RCF}} \cup \Gamma' \rangle$, is completely determined by Γ'. Therefore, we can unabiguously refer to a QoS specification using its set of formulas Γ'. Thus, the QoS specification in Example 4 is Γ'_{Low}.

Example 5 (QoS for POP). The following QoS specifications formalise the costs associated to different activities in the POP protocol of Example 2.

$$\Gamma'_{\text{Chk}} = \{t \leqslant 5, c = 0.5, m = 0\}$$
$$\Gamma'_{\text{Mem}} = \{1 \leqslant t \leqslant 6, c = 0, m \leqslant 64\}$$
$$\Gamma'_{\text{DB}} = \{t \leqslant 3 \implies (\exists x)(0.5 \leqslant x \leqslant 1 \wedge c = t \cdot x), t > 3 \implies c = 10, m \leqslant 5\}$$

Basically, Γ'_{Chk} formalizes the costs associated to the activity of integrity checking a message, Γ'_{Mem} to the activity of a server receiving a message, and Γ'_{DB} to establishing that the monetary cost is fixed if the insertion takes more than three time-units and it is a fraction of the execution time, otherwise. ◇

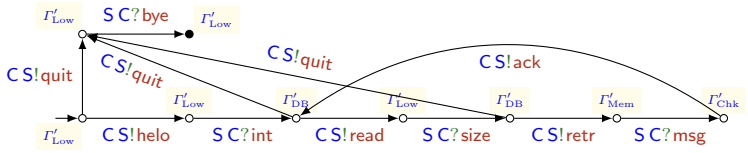
We now extend communicating systems (cf. Section 3) with QoS-specifications.

Definition 4 (QoS-extended CFSMs). *A* QoS-extended CFSM *is a tuple* $M^{QoS} = \langle M, F, \mathit{qos}\rangle$ *where:*

- $M = \langle Q, q_0, \rightarrow\rangle$ *is a CFSM,*
- $F \subseteq Q$ *is a set of* final states *of* M*, and*
- $\mathit{qos} : Q \rightarrow \mathcal{C}$ *maps states of* M *to QoS specifications.*

A QoS-extended communicating system S^{QoS} *is a map* $(M_{\mathsf{A}}^{QoS})_{\mathsf{A}\in\mathcal{P}}$ *assigning an* A*-local QoS-extended CFSM* M_{A}^{QoS} *to each* $\mathsf{A} \in \mathcal{P}$*. A configuration* $\langle \mathfrak{q} \,;\, \mathfrak{b}\rangle$ *of* S^{QoS} *is a* final configuration *if* $\mathfrak{q}(\mathsf{A}) \in F_{\mathsf{A}}$ *for every* $\mathsf{A} \in \mathcal{P}$*.*

Example 6 (QoS-extended CFSMs). An extended CFSM of the POP client in Example 2 with the QoS specifications of Example 5 is as follows:



where the filled state is the only final state. Each state is assigned a QoS specification given in Example 5 according to the following idea. States where the client performs negligible computations are assigned the QoS specification Γ'_{Low}. The remaining states are assigned QoS specifications as follows. The local states where C performs a database insertion (right after receiving an int or size message) and those where C accesses the memory (right before receiving an unread e-mail) are constrained respectively by Γ'_{DB} and Γ'_{Mem}; finally, Γ'_{Chk} constrains the states where C performs an integrity check (right after receiving an unread e-mail). ◇

Notice that Definition 4 requires every state of a CFSM to be assigned a QoS specification. However, in most cases, most states will have the same QoS specification, as it is the case of Γ'_{Low} in Example 6; typically one only has to identify the QoS costs specific to few states.

The semantics of QoS-extended communicating systems is defined in the same way as the semantics of communicating systems. This is a consequence of the fact that QoS specifications do not have any effect on communications.

5 $\mathcal{QL}$: A Dynamic Logic for QoS

To describe QoS properties we introduce $\mathcal{QL}$, a logical language akin *DLTL*.

Definition 5 (QoS formulae). *The* QoS logic $\mathcal{QL}$ *consists of the smallest set of formulae that can be obtained from the following grammar:*

$$\Phi \ ::= \ \top \ \mid \ \psi \ \mid \ \neg\Phi \ \mid \ \Phi \vee \Phi \ \mid \ \Phi \, \mathcal{U}^{\mathsf{G}} \Phi$$

where ψ is a formula in a theory presentation in $\mathcal{C}$, and G is a g-choreography over $\mathcal{P}$ and $\mathcal{M}$ (see Definition 1).

Atomic formulae express constraints over quantitative attributes. Akin *DLTL*, properties of runs are linear temporal formulae where the until operator is indexed with a global choreography G. In essence, the role of G is to restrict the set of runs to be considered for the satisfiability of the until. Global choreographies are suitable for this purpose because they are a declarative and compact way of characterizing the behaviour of asynchronous message-passing systems. The possibility modality $\langle \mathsf{G} \rangle \Phi$ is defined as $\top \mathcal{U}^{\mathsf{G}} \Phi$ and the necessity modality $[\mathsf{G}]\Phi$ is defined (dually) as $\neg\langle \mathsf{G} \rangle \neg\Phi$. Finally, propositional connectives $\wedge$ and $\Longrightarrow$ are defined as usual.

The following example shows how to express non-functional properties of specific runs of the system in $\mathcal{QL}$.

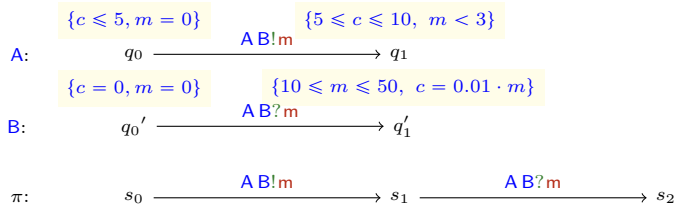
Example 7 (QoS properties of POP protocol). We can use the g-choreographies and the $\mathcal{QL}$ formula below to state that, unless the cost is zero for the first three e-mails read, the cost is bounded by 10 times the CPU time, and the memory consumption is bounded by 5. We define

$$\begin{aligned} \Phi &\equiv [\mathsf{G}_3](c > 0) \implies [\mathsf{G}_3; \mathsf{G}_{\mathsf{msg}}{}^*]\big((c \leqslant t \cdot 10) \wedge (m \leqslant 5)\big) \qquad \text{where} \\ \mathsf{G}_3 &= \mathsf{C}{\to}\mathsf{S}\colon \mathsf{helo}; \mathsf{S}{\to}\mathsf{C}\colon \mathsf{int}; \mathsf{G}_{\mathsf{msg}}; \mathsf{G}_{\mathsf{msg}}; \mathsf{G}_{\mathsf{msg}} \qquad \text{and} \\ \mathsf{G}_{\mathsf{msg}} &= \mathsf{C}{\to}\mathsf{S}\colon \mathsf{read}; \mathsf{S}{\to}\mathsf{C}\colon \mathsf{size}; \mathsf{C}{\to}\mathsf{S}\colon \mathsf{retr}; \mathsf{S}{\to}\mathsf{C}\colon \mathsf{msg}; \mathsf{S}{\to}\mathsf{C}\colon \mathsf{ack} \end{aligned}$$

Intuitively, for Φ to hold, either the first three message retrievals must have zero cost in any run of the system, or on every subsequent message retrieval, the total cost and memory consumption fall within the specified bounds. ◇

A $\mathcal{QL}$ formula (like Φ in Example 7) can be used in quantitative analyses by *aggregating* the values of the QoS attributes along the runs of the system. More precisely, given a run π, our interpretation is that, for each transition $s_i \xRightarrow{\ell_i} s_{i+1}$ of π, the obligations stated in the QoS specification of s_i are met after aggregating QoS information along π from state s_0 up to state s_i. Therefore, a central notion in our framework is that of *aggregation function*. Given a QoS-extended communicating system S, an *aggregation function* $\mathsf{agg}_S : \Delta_S \to \mathcal{C}$ yields a QoS specification capturing the cumulative QoS attributes along a run $\pi \in \Delta_S$ by "summing-up" QoS specifications of participants' local states.

Example 8 (Aggregation). Recall the run π (1) from Section 1:



Let c^q_{A} (resp. c^q_{B}) denote the value of the QoS attribute c in the state q of participant A (resp. B) and likewise for the attribute m. After π, we expect $\max\{m^{q_0}_{\mathsf{A}}, m^{q_1}_{\mathsf{A}}, m^{q'_0}_{\mathsf{B}}, m^{q'_1}_{\mathsf{B}}\}$ and $c^{q_0}_{\mathsf{A}} + c^{q_1}_{\mathsf{A}} + c^{q'_0}_{\mathsf{B}} + c^{q'_1}_{\mathsf{B}}$ to respectively be the memory consumption and the overall monetary cost in s_2. This boils down to aggregate the QoS attributes c and m using the maximization and addition operations, respectively. ◇

Essentially, the aggregation in this case is obtained by (1) instantiating the QoS specification associated to the local state of participants (this is done by renaming attributes as in Example 8); and (2) adding an equation combining all the instances of QoS specifications. The following formula captures this intuition and exemplifies one way in which the aggregation function could be defined.

Example 9 (Aggregation). Let $S = (\langle M_{\mathsf{A}}, \mathsf{qos}_{\mathsf{A}}\rangle)_{\mathsf{A}\in\mathcal{P}}$, we define the aggregation function $\mathsf{agg}_S : \Delta_S \to \mathcal{C}$ to be $\mathsf{agg}_S(\pi) = f(\pi) \cup g(\pi)$ where

$$f(\pi) = \bigcup_{\substack{\mathsf{A}\in\mathcal{P}\\ 0\leqslant i\leqslant n}} \mathsf{qos}_{\mathsf{A}}(\mathfrak{q}_i(\mathsf{A}))^{\mathfrak{q}_i(\mathsf{A})}_{\mathsf{A}} \qquad \text{and} \qquad g(\epsilon) = \left\{ \mathsf{a} = \left(\bigoplus^{\mathsf{a}}_{\mathsf{A}\in\mathcal{P}} \mathsf{a}^{\mathfrak{q}_0(\mathsf{A})}_{\mathsf{A}} \right) \;\middle|\; \mathsf{a} \in \mathsf{Q} \right\}$$

$$g(\pi) = \left\{ \mathsf{a} = \left(\bigoplus^{\mathsf{a}}_{\substack{0\leqslant i<n\\ \mathsf{A}=\mathsf{sbj}(\ell_i)}} \mathsf{a}^{\mathfrak{q}_i(\mathsf{A})}_{\mathsf{A}} \right) \oplus^{\mathsf{a}} \left(\bigoplus^{\mathsf{a}}_{\mathsf{A}\in\mathcal{P}} \mathsf{a}^{\mathfrak{q}_n(\mathsf{A})}_{\mathsf{A}} \right) \;\middle|\; \mathsf{a} \in \mathsf{Q} \right\} \qquad \text{if } \pi \neq \epsilon$$

where $\pi = \langle \mathfrak{q}_0 ; \mathfrak{b}_0\rangle \xrightarrow{\ell_0} \ldots \xrightarrow{\ell_{n-1}} \langle \mathfrak{q}_n ; \mathfrak{b}_n\rangle \in \Delta_S$, and $\Pi^q_{\mathsf{A}} = \{\psi^q_{\mathsf{A}} \mid \psi \in \Pi\}$ for a set of $\mathcal{QL}$ formulae Π, and ψ^q_{A} is obtained by replacing each QoS attribute c with the symbol c^q_{A} in the atomic formula ψ. The intuition is that $f(\pi)$ collects all the QoS specifications of the local states of the participants along the run π, and $g(\pi)$ uses the aggregation operators to calculate the aggregated values of the QoS attributes in the run π. If we apply this aggregation function to the run π in Example 8, we obtain the following:

$$\begin{aligned} f(\pi) = {} & \{c^{q_0}_{\mathsf{A}} \leqslant 5, m^{q_0}_{\mathsf{A}} = 0\} \cup \{5 \leqslant c^{q_1}_{\mathsf{A}} \leqslant 10,\ m^{q_1}_{\mathsf{A}} < 3\} \\ & \cup \{c^{q_0{}'}_{\mathsf{B}} = 0, m^{q_0{}'}_{\mathsf{B}} = 0\} \cup \{10 \leqslant m^{q'_1}_{\mathsf{B}} \leqslant 50,\ c^{q'_1}_{\mathsf{B}} = 0.01 \cdot m^{q'_1}_{\mathsf{B}}\} \\ g(\pi) = {} & \left\{ m = \max\{m^{q_0}_{\mathsf{A}}, m^{q_1}_{\mathsf{A}}, m^{q'_0}_{\mathsf{B}}, m^{q'_1}_{\mathsf{B}}\}, c = c^{q_0}_{\mathsf{A}} + c^{q_1}_{\mathsf{A}} + c^{q'_0}_{\mathsf{B}} + c^{q'_1}_{\mathsf{B}} \right\} \end{aligned}$$

◇

It is important to emphasize that, in our conception, an aggregation function relies on a run of the system as its input. This run inherently encompasses a specific sequential ordering of the actions carried out by the participants. The aggregation operators *max* and + used in Example 8 follow this interpration. As will become clear in Definition 6, this interpretation is sufficient for the purposes of this paper, since it enables $\mathcal{QL}$ to specify temporal QoS properties about runs of the system. However, one might be interested in a different kind of aggregation that is aware of local states that are executed in parallel. This may require some care and possibly to exploit truly-concurrent models, such as pomsets; this is left for future work.

The semantics of our logic is defined in terms of QoS-extended communicating systems.

Definition 6 ($\mathcal{QL}$ semantics). *A run is* terminating *if its last configuration is a final conf. (see Definition 4). Given a QoS-extended communicating system S, an S-*model for a QoS property *Φ is a pair $\langle \pi, \pi' \rangle$, where $\pi \in \Delta_S^\infty$ is a terminating run and $\pi' \in prf(\pi)$, such that $\langle \pi, \pi' \rangle \models_S \Phi$ where the relation $\models_S$ is*

$$
\begin{array}{rcl}
\langle \pi, \pi' \rangle \models_S \Phi & \textit{iff} & \mathit{agg}_S(\pi') \vdash_{RCF} \Phi \quad \textit{if } \Phi \textit{ is an atomic formula} \\
\langle \pi, \pi' \rangle \models_S \neg\Phi & \textit{iff} & \langle \pi, \pi' \rangle \models_S \Phi \textit{ does not hold} \\
\langle \pi, \pi' \rangle \models_S \Phi_1 \vee \Phi_2 & \textit{iff} & \langle \pi, \pi' \rangle \models_S \Phi_1 \textit{ or } \langle \pi, \pi' \rangle \models_S \Phi_2 \\
\langle \pi, \pi' \rangle \models_S \Phi_1 \,\mathcal{U}^{\mathsf{G}}\, \Phi_2 & \textit{iff} & \textit{there exists } \pi'' \textit{ such that } \pi'\pi'' \in prf(\pi) \textit{ and } \mathcal{L}[\pi''] \in \hat{\mathcal{L}}[\mathsf{G}], \\
 & & \textit{satisfying } \langle \pi, \pi'\pi'' \rangle \models_S \Phi_2 \textit{ and, for all } \pi''' \in prf(\pi''), \\
 & & \textit{if } \pi''' \neq \pi'' \textit{ then } \langle \pi, \pi'\pi''' \rangle \models_S \Phi_1.
\end{array}
$$

A QoS property Φ is satisfiable *in S if there exists a terminating run $\pi \in \Delta_S^\infty$ such that $\langle \pi, \epsilon \rangle \models_S \Phi$, and it is* valid *(denoted as $\models_S \Phi$) if $\langle \pi, \epsilon \rangle \models_S \Phi$ for all terminating runs $\pi \in \Delta_S^\infty$.*

Negation and disjunction are handled in the standard way. The definition of the until operator is similar to the standard operator: Φ_2 must hold at some point in the future, i.e., $\pi'\pi''$ and Φ_1 must hold up to that point; the key difference is that the satisfaction of Φ_2 is restricted to runs where the extension π'' is in $\hat{\mathcal{L}}[\mathsf{G}]$. Finally, atomic formulae are handled by obtaining the aggregated QoS of the accumulated run π' and using the entailment relation of RCFs.

6 A semidecision procedure for $\mathcal{QL}$

We now establish the semi-decidability of $\mathcal{QL}$ by presenting a k-bounded semi-decision procedure relying on three algorithms: `QSat`, `QModels`, and `QUntil`. The `QSat` algorithm is the main algorithm of the procedure and determines whether a given formula is satisfiable in a given system. It relies on `QModels` to check if there is a run that satisfies the formula which, in turn, uses `QUntil` to handle the $\mathcal{U}$ operator. Let us start by looking at `QSat` defined as

```
1  qSat(Φ, S, k):
2  |  i = 0
3  |  while i ⩽ k do
4  |  |  foreach π ∈ Δ^i_S do
5  |  |  |  if π is a terminating run and qModels(Φ, S, π, ϵ) then
6  |  |  |  |  return true
7  |  |  i = i + 1
8  |  return false
```

Basically, `qSat` enumerates all the runs of S up to a given bound k and checks whether any of them satisfies Φ (recall that Δ^i_S is the set of all runs of S with length i). Let us now focus on `qModels`:

```
1  qModels(Φ, S, π, π′):
2  |  switch Φ do
3  |  |  case ⊤ do
4  |  |  |  return true
5  |  |  case ψ do
6  |  |  |  return whether agg_S(π′) ⊢_RCF ψ
7  |  |  case ¬Φ1 do
8  |  |  |  return not qModels(Φ1, S, π, π′)
9  |  |  case Φ1 ∨ Φ2 do
10 |  |  |  return qModels(Φ1, S, π, π′) or qModels(Φ2, S, π, π′)
11 |  |  case Φ1 U^G Φ2 do
12 |  |  |  return qUntil(Φ1, G, Φ2, S, π, π′, ϵ)
```

Following Definition 6, `qModels` recursively inspects the $\mathcal{QL}$ formula. It invokes `qUntil` to handle the $\mathcal{U}$ operator and the decision procedure of the theory of real-closed fields to check the atomic formulae. Let us now look at `qUntil`:

```
1  qUntil(Φ1, G, Φ2, S, π, π′, π″):
2  |  if L[π″] ∈ L̂[G] and qModels(Φ2, S, π, π′π″) then
3  |  |  return true
4  |  else if not qModels(Φ1, S, π, π′π″) then
5  |  |  return false
6  |  else
7  |  |  Let ⟹^ℓ q be the transition such that π′π″ ⟹^ℓ q ∈ prf(π)
   |  |    (takes the first transition in π if π′π″ = ϵ,
   |  |    and it is not defined if π′π″ = π)
8  |  |  if π′π″ = π or L[π″ ⟹^ℓ q] ∉ L[G] then
9  |  |  |  return false
10 |  |  else
11 |  |  |  return qUntil(Φ1, G, Φ2, S, π, π′, π″ ⟹^ℓ q)
```

This procedure takes care of searching for a witness of the existential in the semantics of $\mathcal{U}$ by starting in the current prefix π' and following the transitions of π. According to Definition 6, `qUntil` searches for a witness of the existential part of $\mathcal{U}$. It takes as parameters the complete run π, the prefix π' at which the $\mathcal{U}$ is being evaluated, and the current extension π'' that is used to search for the witness. If π'' is enough to reach a verdict, the algorithm returns true or false accordingly (Lines 3 and 5). Otherwise, it tries to extend π'' by borrowing the next transition of π (Line 7). If such extension exists and is a candidate for being in the language of G, the algorithm recursively calls itself with the extended prefix (Line 11). Hereafter, we fix a QoS-extended communicating system S.

Theorem 1 (`qSat` is sound and k-bounded complete). *Given a QoS formula $\Phi \in \mathcal{QL}$ and a bound k, `qSat`(Φ, S, k) returns **true** iff there exists $\pi \in \Delta^i_S$ such that $\langle \pi, \epsilon \rangle \models_S \Phi$, for some $i \leqslant k$.*

The soundness of qSat immediately follows from the soundness of qModels (established in Lemma 1 below) which, in turn, relies on the soundness and completeness of qUntil (cf. Lemmas 2 and 3, respectively). This guarantees that the call to qModels in Line 5 of qSat returns true iff the run π satisfies Φ. Note that qSat is not guaranteed to be complete due to the bound k.

Lemma 1 (qModels is sound and complete). *Given a QoS formula $\Phi \in \mathcal{QL}$ and runs $\pi, \pi' \in \Delta_S$, where $\pi' \in prf(\pi)$,* qModels(Φ, S, π, π') *returns* ***true*** *iff* $\langle \pi, \pi' \rangle \models_S \Phi$.

Proof. By structural induction on Φ. If Φ is $\top$, the result follows trivially. If Φ is an atomic formula, the algorithm computes the aggregation over the run π' (Line 6) and invokes the decision procedure of RCFs to check whether $\mathtt{agg}_S(\pi')$ entails Φ in the theory of real closed fields. If Φ is $\Phi_1 \vee \Phi_2$, the algorithm perform two recursive calls and returns true iff either $\langle \pi, \pi' \rangle \models_S \Phi_1$ or $\langle \pi, \pi' \rangle \models_S \Phi_2$. If Φ is $\Phi_1 \, \mathcal{U}^{\mathsf{G}} \, \Phi_2$, the algorithm returns true iff qUntil$(\Phi_1, \mathsf{G}, \Phi_2, S, \pi, \pi', \epsilon)$ returns true. By Lemmas 2 and 3 this is equivalent to $\langle \pi, \pi' \rangle \models_S \Phi_1 \, \mathcal{U}^{\mathsf{G}} \, \Phi_2$. □

We now prove the soundness and completeness of qUntil.

Lemma 2 (qUntil is sound). *Given a QoS formula $\Phi_1, \Phi_2 \in \mathcal{QL}$, a g-choreography G, and runs π, π', π'' such that*

a) $\pi'\pi'' \in prf(\pi)$ and $\pi \in \Delta_S$, and
b) for all $\pi''' \in prf(\pi'')$, if $\pi''' \neq \pi''$ then $\langle \pi, \pi'\pi''' \rangle \models_S \Phi_1$

if qUntil$(\Phi_1, \mathsf{G}, \Phi_2, S, \pi, \pi', \pi'')$ *returns* ***true*** *then* $\langle \pi, \pi' \rangle \models_S \Phi_1 \, \mathcal{U}^{\mathsf{G}} \, \Phi_2$.

Proof. The call to qUntil$(\Phi_1, \mathsf{G}, \Phi_2, S, \pi, \pi', \pi'')$ either reaches Line 3 or it reaches Line 11 and the recursive call returns true. In the first case, we know $\mathcal{L}[\pi''] \in \hat{\mathcal{L}}[\mathsf{G}]$ and that qModels$(\Phi_2, S, \pi, \pi'\pi'')$ returned true. By Lemma 1 it follows that $\langle \pi, \pi'\pi'' \rangle \models_S \Phi_2$. Together with hypotheses a) and b) the conditions of the semantics of the formula $\Phi_1 \, \mathcal{U}^{\mathsf{G}} \, \Phi_2$ (see Definition 6) are met. In the case of reaching Line 11 we know the recursive call qUntil$(\Phi_1, \mathsf{G}, \Phi_2, S, \pi, \pi', \pi'' \overset{\ell}{\Longrightarrow} q)$ returned true. Conditions a) and b) applied to the input of the recursive call holds because of the way transition $\overset{\ell}{\Longrightarrow} q$ was chosen and the fact that condition on Line 4 returned false. Therefore, we can take the output of the recursive call to satisfy Lemma 2 as an inductive hypothesis and conclude $\langle \pi, \pi' \rangle \models_S \Phi_1 \, \mathcal{U}^{\mathsf{G}} \, \Phi_2$. □

Lemma 3 (Completeness of qUntil). *Given a QoS formula $\Phi_1, \Phi_2 \in \mathcal{QL}$, a g-choreography G, and runs π, π', π'' such that*

a) $\pi'\pi'' \in prf(\pi)$ and $\pi \in \Delta_S$, and
b) for all $\pi''' \in prf(\pi'')$, if $\pi''' \neq \pi''$ then either $\mathcal{L}[\pi'''] \notin \hat{\mathcal{L}}[\mathsf{G}]$ or $\langle \pi, \pi'\pi''' \rangle \not\models_S \Phi_2$

if qUntil$(\Phi_1, \mathsf{G}, \Phi_2, S, \pi, \pi', \pi'')$ *returns* ***false*** *then* $\langle \pi, \pi' \rangle \not\models_S \Phi_1 \, \mathcal{U}^{\mathsf{G}} \, \Phi_2$

Proof. The call to $\texttt{qUntil}(\Phi_1, \mathsf{G}, \Phi_2, S, \pi, \pi', \pi'')$ reaches either Line 5, Line 9 or it reaches Line 11 and the recursive call returns false. In all cases, by condition b) we know that no prefix of π'' could be witness of the existential in the semantics of $\Phi_1 \; \mathcal{U}^{\mathsf{G}} \; \Phi_2$ (see Definition 6) because it would need to both be in $\hat{\mathcal{L}}[\mathsf{G}]$ and satisfy Φ_2. Run π'' itself cannot be the witness for the same reasons due to the fact that condition in Line 2 was not met. Which means either $\mathcal{L}[\pi''] \notin \hat{\mathcal{L}}[\mathsf{G}]$ or $\texttt{qModels}(\Phi_2, S, \pi, \pi'\pi'')$ returned false, therefore, using Lemma 1, either $\mathcal{L}[\pi''] \notin \hat{\mathcal{L}}[\mathsf{G}]$ or $\langle \pi, \pi'\pi'' \rangle \not\models_S \Phi_2$. The only remaining possibility is for the witness to be a $\pi^\star$ such that $\pi'' \in \mathit{prf}(\pi^\star)$ and $\pi^\star \neq \pi''$. In the case of reaching Line 5, we know that $\texttt{qModels}(\Phi_1, S, \pi, \pi'\pi'')$ returned false. By Lemma 1 it follows that $\langle \pi, \pi'\pi'' \rangle \not\models_S \Phi_1$. Therefore, extension $\pi^\star$ couldn't be a witness for the existential in the semantics of $\Phi_1 \; \mathcal{U}^{\mathsf{G}} \; \Phi_2$. In the case of reaching Line 9, candidate extension $\pi^\star$ does not exist or it is not in $\hat{\mathcal{L}}[\mathsf{G}]$. In the case of reaching Line 11, we know that $\texttt{qUntil}(\Phi_1, \mathsf{G}, \Phi_2, S, \pi, \pi', \pi'' \overset{\ell}{\Rightarrow} q)$ returned false. Notice that conditions a) and b) applied to the input of the recursive calls holds because of the way transition $\overset{\ell}{\Rightarrow} q$ was chosen and that condition in Line 2 was not met. Therefore, we can take the output of the recursive calls to satisfy Lemma 3 as an inductive hypothesis and conclude that $\langle \pi, \pi' \rangle \not\models \Phi_1 \; \mathcal{U}^{\mathsf{G}} \; \Phi_2$. □

Notice that the proof for Lemma 1 uses Lemma 3 and Lemma 2, and that the proofs for Lemma 3 and Lemma 2 use Lemma 1. This does not undermine the soundness of the proofs because the lemmas are always (inductively) applied on smaller $\mathcal{QL}$ formulas. Now that the soundness and completeness of qModels and qUntil is established, it remains to show their termination. Termination follows from the fact that both the number of logical operators in Φ and the number of transitions in π are finite. The first guarantees qModels eventually reaches a base case and the second guarantees qUntil eventually reaches a base case. Finally, the base case in qModels, computing aggregation and checking entailment in the theory of real closed fields, terminates due to the decidability of RCFs [17].

6.1 A bounded model-checking approach for $\mathcal{QL}$

Previous results allow for a straightforward bounded model-checking approach for $\mathcal{QL}$. Like for other model-checking procedures for a language that admits negation, qSat can be used to check validity of a $\mathcal{QL}$ formula in a system S by checking the satisfiability of the negated formula. This constitutes a counterexample-finding procedure for $\mathcal{QL}$. The caveat is that qSat is a k-bounded semidecision procedure rather than a decision procedure. However, restricting to $\mathcal{QL}^-$, namely $\mathcal{QL}$ formulae that do not contain the $*$ operator in their choregraphies, we can find finite models of satisfiable formulae of $\mathcal{QL}^-$ (cf. Theorem 2). Thus, qSat can serve as a decision procedure for $\mathcal{QL}^-$, if one computes a suitable bound k.

Theorem 2 (Finite model property of $\mathcal{QL}^-$). *Given a QoS formula $\Phi \in \mathcal{QL}^-$, and runs $\pi \in \Delta_S^\infty$, $\pi' \in \Delta_S$ such that $\pi' \in \mathit{prf}(\pi)$. If $\langle \pi, \pi' \rangle \models_S \Phi$ then there exists a finite run $\pi^- \in \Delta_S$ such that $\pi^- \in \mathit{prf}(\pi)$ and $\langle \pi^-, \pi' \rangle \models_S \Phi$.*

Proof. By structural induction on Φ. If Φ is $\top$ or an atomic formula, take $\pi^- = \pi'$. If Φ is $\Phi_1 \vee \Phi_2$, we have that either $\langle \pi, \pi' \rangle \models_S \Phi_1$ or $\langle \pi, \pi' \rangle \models_S \Phi_2$. By inductive hypothesis, either $\langle \pi_1^-, \pi' \rangle \models_S \Phi_1$ or $\langle \pi_2^-, \pi' \rangle \models_S \Phi_2$ for some finite $\pi_1^-, \pi_2^- \in prf(\pi)$. Therefore, either $\langle \pi_1^-, \pi' \rangle \models_S \Phi_1 \vee \Phi_2$ or $\langle \pi_2^-, \pi' \rangle \models_S \Phi_1 \vee \Phi_2$.

If Φ is $\Phi_1 \; \mathcal{U}^{\mathsf{G}} \; \Phi_2$, by Definition 6 we have there exists π'' such that $\pi'\pi'' \in prf(\pi)$ and $\mathcal{L}[\pi''] \in \mathcal{L}[\mathsf{G}]$, and $\langle \pi, \pi'\pi'' \rangle \models_S \Phi_2$, and for all $\pi''' \in prf(\pi'')$, if $\pi''' \neq \pi''$ then $\langle \pi, \pi'\pi''' \rangle \models_S \Phi_1$. If we apply the inductive hypothesis to Φ_1 and Φ_2, we have there exists π'' such that $\pi'\pi'' \in prf(\pi)$ and $\mathcal{L}[\pi''] \in \mathcal{L}[\mathsf{G}]$, and $\langle \pi_2^-, \pi'\pi'' \rangle \models_S \Phi_2$ for some $\pi_2^- \in prf(\pi)$, and for all $\pi''' \in prf(\pi'')$, if $\pi''' \neq \pi''$ then $\langle \pi_1^-, \pi'\pi''' \rangle \models_S \Phi_1$ for some $\pi_1^- \in prf(\pi)$. Notice that since G is $*$-free, run π'' in the language $\mathcal{L}[\mathsf{G}]$ is necessarily finite and so is the number of quantified runs π'''. Therefore, the number of runs π_1^- involved in the previous statement is finite, and there is a maximum among their lengths, so we can take π^- as the longest between π_2^- and runs π_1^-. Since π_2^- and all the π_1^- are prefixes of π, then they will also be prefixes of π^-, and therefore we have the conditions to conclude $\langle \pi^-, \pi' \rangle \models_S \Phi_1 \; \mathcal{U}^{\mathsf{G}} \; \Phi_2$.

If the outermost operator in Φ is $\neg$, we need to consider al the possible cases for the immediate subformula of Φ. If Φ is $\neg\psi$ with ψ atomic formula, we have that $\langle \pi, \pi' \rangle \not\models_S \psi$. It follows that $\langle \pi', \pi' \rangle \not\models_S \psi$ and we can take $\pi^- = \pi'$. If Φ is $\neg(\Phi_1 \vee \Phi_2)$, we have that $\langle \pi, \pi' \rangle \not\models_S \Phi_1 \vee \Phi_2$. It follows that $\langle \pi, \pi' \rangle \not\models_S \Phi_1$ and $\langle \pi, \pi' \rangle \not\models_S \Phi_2$. By inductive hypothesis, there exists $\pi_1^- \in \Delta_S$ such that $\pi_1^- \in prf(\pi)$ and $\langle \pi_1^-, \pi' \rangle \not\models_S \Phi_1$ and there exists $\pi_2^- \in \Delta_S$ such that $\pi_2^- \in prf(\pi)$ and $\langle \pi_2^-, \pi' \rangle \not\models_S \Phi_2$. It is enough to take π^- as the longest between π_1^- and π_2^-. If Φ is $\neg(\Phi_1 \; \mathcal{U}^{\mathsf{G}} \; \Phi_2)$, we have that $\langle \pi, \pi' \rangle \not\models_S \Phi_1 \; \mathcal{U}^{\mathsf{G}} \; \Phi_2$. Therefore, for all π'' such that $\pi'\pi'' \in prf(\pi)$, if $\mathcal{L}[\pi''] \in \mathcal{L}[\mathsf{G}]$, and $\langle \pi, \pi'\pi'' \rangle \models_S \Phi_2$, then there exists $\pi''' \in prf(\pi'')$, with $\pi''' \neq \pi''$ such that $\langle \pi, \pi'\pi''' \rangle \models_S \neg\Phi_1$. If we apply the inductive hypothesis to Φ_2 and $\neg\Phi_1$, we have that for all π'' such that $\pi'\pi'' \in prf(\pi)$, if $\mathcal{L}[\pi''] \in \mathcal{L}[\mathsf{G}]$, and $\langle \pi_2^-, \pi'\pi'' \rangle \models_S \Phi_2$ for some $\pi_2^- \in \Delta_S$ with $\pi_2^- \in prf(\pi)$, then there exists $\pi''' \in prf(\pi'')$, with $\pi''' \neq \pi''$ such that $\langle \pi_1^-, \pi'\pi''' \rangle \models_S \neg\Phi_1$ for some $\pi_1^- \in \Delta_S$ with $\pi_1^- \in prf(\pi)$. Notice that since G is $*$-free, any run in the language $\mathcal{L}[\mathsf{G}]$ is necessarily finite. Therefore, there is a maximum among the lengths of the runs π_2^-, and we can take π^- as the longest between π_2^- and π_1^-. □

The proof of Thm. 2 suggests how to compute a bound on the length of the model π^- that satisfies the lemma. That is, recursively in the structure of Φ by taking the maximum between the bounds of the immediate subformulae and, for the case of the $\mathcal{U}^{\mathsf{G}}$ operator, the length of the longest run in $\mathcal{L}[\mathsf{G}]$. This bound can be used as k in `QSAT` to obtain a full decision procedure for $\mathcal{QL}^-$. Searching for counterexamples of an arbitrary formula $\Phi \in \mathcal{QL}$ up to a bounded number of unfoldings of $*$ is equivalent to searching for counterexamples in a formula $\hat{\Phi}$ in $\mathcal{QL}^-$ where each $*$ has been replaced by a finite number of unfoldings. Which means that the bound computed for $\mathcal{QL}^-$ can be used to search for counterexamples in $\mathcal{QL}$. Notice that `QSAT` can be easily extended to return the run that satisfies the formula, if there is one. Which can help identify the source of QoS formula violations when used as a counterexample-finding procedure.

7 Conclusions

We presented a framework for the design and analysis of QoS-aware distributed message-passing systems using choreographies and a general model of QoS. We tackle this problem by: 1) abstractly representing QoS attributes as symbols denoting real values, whose behaviour is completely captured by a decidable RCFs theory, 2) extending the choreographic model of CFSM by associating QoS specifications to each state of the machine, 3) introducing $\mathcal{QL}$, a logic based on *DLTL*, for expressing QoS properties with a straightforward satisfaction relation based on runs of communicating systems, and 4) giving a semi-decision procedure for $\mathcal{QL}$ and defining a decidable fragment $\mathcal{QL}^{-}$ that allowed us to give a bounded model-checking procedure for the full logic. A prototype implementation of our procedure is under development. It relies on the SMT solver Z3 [34] for the satisfiability of the QoS constraints in atomic formulae and on ChorGram [35,36] for the semantics of g-choreographies and CFSMs. An interesting by-product of our framework is that it could be used for the monitoring of local computations to check at run-time if they stay in the constraint of QoS specifications. If static guarantees on QoS specifications are not possible, run-time monitors can be easily attained by adapting techniques for monitor generation from behavioural types [37,38].

We identify two further main future research directions. On the one hand, there is the theoretical question of whether $\mathcal{QL}$ is decidable or not. In this respect, the similarity of $\mathcal{QL}$ with *DLTL* (cf. Section 2) hints towards an affirmative answer suggesting that the problem can be resolved to emptyness of Büchi automata [39] corresponding to $\mathcal{QL}$ formulae. However, the decidability of $\mathcal{QL}$ is not so easy to attain. In general, a $\mathcal{QL}$ formula may yield an infinite state space due to instantiation of QoS attributes. On the other hand, the usability of the framework could be improved through two extensions of $\mathcal{QL}$ and a less demanding way of modeling QoS-extended communicating systems. The first extension of $\mathcal{QL}$ are *selective* aggregation, enabling the aggregation of QoS attributes only for some specific states of runs. This can be done by extending the grammar of g-choreographies given in Definition 1 with an extra production of the shape $\mathsf{G} ::= \cdots \mid [\mathsf{G}]$, "bracketing" the part of the choreography relevant for the aggregation. Notice that the run still has to match the whole choreography. A second extension of $\mathcal{QL}$ is the introduction of *wildcards* as a mechanism to "ignore" a subchoreography. Syntactically, it can be represented by, once again, extending the grammar given in Definition 1 with an extra production, with shape $\mathsf{G} ::= \cdots \mid \_$, where $\_$ is interpreted as a wildcard and plays the role of matching any possible g-choreography. In this case, the shape of the part of the run that matches the wildcard is disregarded but attributes are aggregated along the whole run. Finally, a less demanding way of modeling QoS-extended systems could be achieved by extending g-choreographies with QoS specifications annotating specific interactions and extending the projection of g-choregraphies into CFSMs taking into account such annotations.

References

1. Obj. Mgmt. Group: Business Process Model and Notation http://www.bpmn.org.
2. Bonér, J.: Reactive Microsystems - The Evolution Of Microservices At Scale. O'Reilly (2018)
3. Frittelli, L., Maldonado, F., Melgratti, H.C., Tuosto, E.: A choreography-driven approach to apis: The opendxl case study. In [4], 107–124.
4. Bliudze, S., Bocchi, L.: Proc. of COORDINATION - 22nd Intl. Conf. IFIP WG 6.1, Valletta, Malta, June 15-19, 2020. Vol. 12134 of LNCS, Springer (2020).
5. Autili, M., Inverardi, P., Tivoli, M.: Automated synthesis of service choreographies. IEEE Softw. **32**(1) (2015) 50–57.
6. World Wide Web Consortium: Web services description language (wsdl) version 2.0 part 1: Core language. On-line Available at https://www.w3.org/TR/wsdl20/.
7. Ivanović, D., Carro, M., Hermenegildo, M.V.: A constraint-based approach to quality assurance in service choreographies. In Liu, C., et.al. eds.: Proc. of SOC (2012) 252–267.
8. Kattepur, A., Georgantas, N., Issarny, V.: Qos analysis in heterogeneous choreography interactions. In Basu, S., et.al. eds.: Proc. of SOC (2013) 23–38.
9. Güdemann, M., Poizat, P., Salaün, G., Ye, L.: Verchor: A framework for the design and verification of choreographies. IEEE Trans. Serv. Comput. **9**(4) (2016) 647–660.
10. Autili, M., Salle, A.D., Gallo, F., Pompilio, C., Tivoli, M.: Chorevolution: Automating the realization of highly-collaborative distributed applications. In: Proc. of COORDINATION - 21st Intl. Conf. IFIP WG 6.1, 2019. Springer (2019) 92–108.
11. Bocchi, L., Melgratti, H.C., Tuosto, E.: On resolving non-determinism in choreographies. Log. Methods Comput. Sci. **16**(3) (2020).
12. Basu, S., Bultan, T., Ouederni, M.: Deciding choreography realizability. In Field, J., et.al. eds.: Proc. of the 39th ACM SIGPLAN-SIGACT POPL 2012. ACM (2012) 191–202.
13. Hüttel, H., Lanese, I., Vasconcelos, V.T., Caires, L., Carbone, M., Deniélou, P., Mostrous, D., Padovani, L., Ravara, A., Tuosto, E., Vieira, H.T., Zavattaro, G.: Foundations of session types and behavioural contracts. ACM Comput. Surv. **49**(1) (2016) 3:1–3:36.
14. Brand, D., Zafiropulo, P.: On communicating finite-state machines. Journal of the ACM **30**(2) (1983) 323–342.
15. Henriksen, J.G., Thiagarajan, P.: Dynamic linear time temporal logic. Annals of Pure and Applied Logic **96**(1–3) (1999) 187–207.
16. Tuosto, E., Guanciale, R.: Semantics of global view of choreographies. Journal of Logical and Algebraic Methods in Programming **95** (2018) 17–40.
17. Tarski, A.: A decision method for elementary algebra and geometry. Memorandum RM-109, RAND Corporation (1951).
18. Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I.: Software architecture optimization methods: A systematic literature review. IEEE Transactions on Software Engineering **39** (2013) 658–683.
19. Hayyolalam, V., Pourhaji Kazem, A.A.: A systematic literature review on QoS-aware service composition and selection in cloud environment. Journal of Network and Comp. Applications **110** (Timothy C. May 2018) 52–74.
20. Giachino, E., de Gouw, S., Laneve, C., Nobakht, B.: Statically and dynamically verifiable SLA metrics. In Ábrahám, E., et.al. eds.: Theory and Practice of Formal Methods - Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday. Vol. 9660 of LNCS, Springer (2016) 211–225.

21. Rosenthal, K.: Quantales and Their Applications. Vol. 234 of Pitman Research Notes in Mathematics Series. Longman Scientific & Technical (1990).
22. Buscemi, M.G., Montanari, U.: Cc-pi: A constraint-based language for specifying service level agreements. In DeNicola, R., ed.: Proc. of 16th ESOP. Vol. 4421 of LNCS, Springer (2007) 18–32.
23. Lluch-Lafuente, A., Montanari, U.: Quantitative μ-calculus and CTL based on constraint semirings. In Proc. of QAPL, 2004. Elec. Notes in Theo. Comp. Sci. **112** (2005) 37–59.
24. De Nicola, R., Ferrari, G., Montanari, U., Pugliese, R., Tuosto, E.: A process calculus for QoS-aware applications. In Jacquet, J., et.al. eds.: Proc. of COORDINATION, 2005. Vol. 3454 of LNCS, Springer (2005) 33–48.
25. Martinez Suñé, A.E., Lopez Pombo, C.G.: Automatic quality-of-service evaluation in service-oriented computing. In Nielson, H.R., et.al. eds.: Proc. of COORDINATION - 21st Intl. Conf. IFIP WG 6.1. Vol. 11533 of LNCS, Springer (2019) 221–236.
26. Bocchi, L., Honda, K., Tuosto, E., Yoshida, N.: A theory of design-by-contract for distributed multiparty interactions. In Gastin, P.,et.al. eds.: Proc. of 21th CONCUR 2010. Vol. 6269 of LNCS, Springer (2010) 162–176.
27. Vissani, I., Lopez Pombo, C.G., Tuosto, E.: Communicating machines as a dynamic binding mechanism of services. In Gay, D., et.al. eds.: Proc. of PLACES. Vol. 203 of Elect. Proc. in Theo. Comp. Sci.. (April 2016) 85–98.
28. Pnueli, A.: The temporal semantics of concurrent programs. Theo. Comp. Sci. **13**(1) (1981) 45–60.
29. Pratt, V.R.: Semantical consideration on floyd-hoare logic. In Carlyle, et.al. eds.: Proc. of 17th SFCS, IEEE Computer Society (1976) 109–121.
30. Vardi, M.Y.: The Rise and Fall of LTL: Invited Talk at the 2nd. Games, Automata, Logics and Formal Verification. Elec. Proc. in Theo. Comp. Sci. **54** (2011).
31. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Proc. of 23rd IJCAI, 2013, AAAI Press (2013) 854–860.
32. Post Office Protocol: Version 2. RFC 937 (1985).
33. Pratt, V.R.: Modeling concurrency with partial orders. Intl. Journal Parallel Programming **15**(1) (1986) 33–71.
34. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In Ramakrishnan, et.al. eds.: Proc. of 14th TACAS, 2008. Vol. 4963 of LNCS, Springer (2008) 337–340.
35. Coto, A., Guanciale, R., Tuosto, E.: Choreographic development of message-passing applications - A tutorial. In [4], 20–36.
36. Coto, A., Guanciale, R., Lange, J., Tuosto, E.: **ChorGram**: tool support for choreographic deveelopment. Available at https://bitbucket.org/eMgssi/chorgram/src/master/ (2015).
37. Francalanza, A., Mezzina, C.A., Tuosto, E.: Towards choreographic-based monitoring. In: Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405. Vol. 12070 of LNCS, Springer (2020) 128–150.
38. Bocchi, L., Chen, T., Demangeon, R., Honda, K., Yoshida, N.: Monitoring networks through multiparty session types. Theor. Comput. Sci. **669** (2017) 33–58.
39. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Proc. of the Intl. Congress on Logic, Method, and Philosophy of Science, Stanford, CA, USA, Stanford University, Stanford University Press (1962) 1–11.
40. Hilbert, D., Ackermann, W.: Principles of mathematical logic. Chelsea publishing company (1928).
41. Biere, A., Heule, M., Van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability: Second Edition. IOS Press (February 2021)