IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

Contract-based discovery of Web services modulo simple orchestrators

This is the author's manuscript

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/70905 since 2017-11-22T15:45:36Z

Published version:

DOI:10.1016/j.tcs.2010.05.002

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

19 March 2024



UNIVERSITÀ DEGLI STUDI DI TORINO

This Accepted Author Manuscript (AAM) is copyrighted and published by Elsevier. It is posted here by agreement between Elsevier and the University of Turin. Changes resulting from the publishing process - such as editing, corrections, structural formatting, and other quality control mechanisms - may not be reflected in this version of the text. The definitive version of the text was subsequently published in THEORETICAL COMPUTER SCIENCE, 411(37), 2010, 10.1016/j.tcs.2010.05.002.

You may download, copy and otherwise use the AAM for non-commercial purposes provided that your license is limited by the following restrictions:

(1) You may use this AAM for non-commercial purposes only under the terms of the CC-BY-NC-ND license.

(2) The integrity of the work and identification of the author, copyright owner, and publisher must be preserved in any copy.

(3) You must attribute this AAM in the following format: Creative Commons BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/deed.en), 10.1016/j.tcs.2010.05.002

The definitive version is available at:
http://linkinghub.elsevier.com/retrieve/pii/S0304397510002720

Contract-based discovery of Web services modulo simple orchestrators

Luca Padovani

Istituto di Scienze e Tecnologie dell'Informazione, Università di Urbino
Piazza della Repubblica 13, 61029 Urbino, Italy

Abstract

Web services are distributed processes with a public description of their behavior, or *contract*. The availability of repositories of Web service descriptions enables interesting forms of dynamic Web service discovery, such as searching for Web services exposing a specified contract. This calls for a formal notion of contract equivalence satisfying two contrasting goals: being as coarse as possible so as to favor Web services reuse, and guaranteeing successful client/service interaction.

We study an equivalence relation that achieves both goals under the assumption that client/service interactions may be mediated by *simple orchestrators*. In the framework we develop, orchestrators play the role of proofs (in the Curry-Howard sense) justifying an equivalence relation between contracts. This makes it possible to automatically synthesize orchestrators out of Web services contracts.

Key words: Web service discovery, behavioral contracts, orchestration, CCS, testing semantics

1 Introduction

Web services are distributed processes equipped with a public description of their interface. Such description typically includes the type – or *schema* – of messages exchanged with the service, the *operations* provided by the service [13], and also the behavior – or *contract* – supported by the service [3, 1]. The description is made public by registering the service in one or more Web service repositories [4, 15, 6, 38] that can be queried for discovering services providing a particular contract. This calls for a formalization of the contract

Email address: padovani@sti.uniurb.it (Luca Padovani).

language and of a characterization of the contract equivalence relation. Indeed, naive notions of contract equivalence, including syntactic or structural equivalences, are too strict to be useful in this context. The aim of this paper is the definition of a notion of contract equivalence – and more specifically of a *subcontract relation* – that can be proficiently used for the discovery of *all* and *only* those Web services that *can* satisfy a given client. Here, "only" means that Web services whose contracts are deemed equivalent should be compatible, namely they should satisfy the same clients; "all" means that the equivalence relation should be as coarse as possible, so a to maximize the search space and favor Web service reuse; "can" means that we should tolerate a certain amount of incompatibility between Web services whose contracts are deemed equivalent, provided that there is a sufficiently simple (i.e. automatic) way of avoiding such incompatibilities.

We express contracts using a fragment of CCS [18] with two choice operators ($+$ for external choice and $\oplus$ for internal choice) without relabeling, restriction, and parallel composition. For instance, the contract $\sigma = a.c.(\overline{b} \oplus \overline{d})$ describes a service that accepts two messages a and c (in this order) and then internally decides whether to send back either $\overline{b}$ or $\overline{d}$. The contract $\rho = \overline{a}.\overline{c}.(b.\mathsf{e} + d.\mathsf{e})$ describes a client that sends two messages a and c (in this order), then waits for either the message b or the message d, and finally terminates (e denotes successful termination). The compliance relation $\rho \dashv \sigma$ tells us that the client ρ is satisfied by the service σ, because every possible interaction between ρ and σ leads to the client terminating successfully. This is not true for ρ and $\sigma' = a.c.(\overline{b} \oplus \overline{c})$, because the service with contract σ' may internally decide to send a message $\overline{c}$ that the client is never willing to accept, hence $\rho \not\dashv \sigma'$. The subcontract relation $\sigma \preceq \tau$, where $\tau = a.c.\overline{b}$, tells us that every client satisfied by σ (including ρ) is also satisfied by τ. This is because τ is more deterministic than σ.

Formal notions of compliance and subcontract relation may be used for implementing contract-based query engines. The query for services that satisfy ρ is answered with the set $\mathscr{Q}_1(\rho) = \{\sigma \mid \rho \dashv \sigma\}$. The complexity of running this query grows with the number of services stored in the repository. A better strategy is to compute the *dual contract* of ρ, denoted by $\rho^\perp$, which represents the canonical service satisfying ρ (that is $\rho \dashv \rho^\perp$) and then answering the query with the set $\mathscr{Q}_2(\rho) = \{\sigma \mid \rho^\perp \preceq \sigma\}$. If $\rho^\perp$ is the $\preceq$-smallest service that satisfies ρ, we have $\mathscr{Q}_1(\rho) = \mathscr{Q}_2(\rho)$, namely we are guaranteed that no service is mistakenly excluded. The advantage of this approach is that $\preceq$ can be precomputed when services are registered in the repository, and the query engine needs only scan through the $\preceq$-minimal contracts. Furthermore, the definition of a formal theory of contracts and of a notion of contract equivalence finds useful applications also outside the scope of Web service discovery: it may help and drive the development of new Web services, as well as supporting maintenance and refactoring of existing ones.

When looking for a suitable theory defining $\dashv$ and $\preceq$, the testing framework [17, 26] and the *must* preorder seem particularly appealing: clients are tests, compliance encodes the passing of a test, and the subcontract relation is the liveness-preserving preorder induced by the compliance relation. Among the characterizing laws of the *must* preorder is $\sigma \oplus \tau \preceq \sigma$, namely it is always safe to replace a (service with) contract $\sigma \oplus \tau$ with a more deterministic one. Unfortunately, the *must* preorder excludes many other relations that are desirable in the context of Web service discovery. For example, a service with contract $a + b$ cannot replace a service with contract a despite the fact that, intuitively, $a + b$ offers more options than just a. The reason is that the client $\rho' = \overline{a}.\mathsf{e} + \overline{b}.c.\mathsf{e}$ complies with a simply because no interaction on b is possible, whereas it can get stuck when interacting with $a + b$ because such service does not offer $\overline{c}$ after b. The relation $a \preceq a + b$ characterizes so-called extension or implementation refinements [21] and it is a well-known fact that it is generally unsafe (it may cause deadlocks). In the context of search engines it is natural to allow this kind of refinement, since it enables the retrieval of more precise services from partial specifications (after all, this is the main task of every modern search engine, even textual ones). As another example of refinement that is not allowed by the *must* preorder, the client $\rho'' = \overline{c}.\overline{a}.(b.\mathsf{e} + d.\mathsf{e})$ fails to interact successfully with σ above because it sends the messages $\overline{a}$ and $\overline{c}$ in the wrong order. The rationale for enabling this kind of refinements is the same that has driven the research on type isomorphisms for function libraries [36, 20], where permutation of arguments and currification are the main characterizing morphisms. In our context, it makes sense to retrieve services whose contract differs from the searched one solely because messages are exchanged in a different order that does not disrupt message dependencies.

In order to accommodate all of the above desiderata, we propose an extension of the classical testing framework where we assume that client and service interact under the supervision of an *orchestrator*. In the Web services domain, an orchestrator coordinates in a centralized way two (or more) interacting parties so as to achieve a specific goal, in our case to guarantee client satisfaction. The orchestrator cannot affect the internal decisions of client and service, but it can affect the way they try to synchronize with each other. In our framework an orchestrator is a *bounded, directional, controlled buffer*: the buffer is bounded in that it can store a finite amount of messages; the buffer is directional in that it distinguishes messages sent to the client from messages sent to the service; the buffer is controlled by orchestration actions. *Asynchronous actions* have either the form $\langle \alpha, \varepsilon \rangle$ or $\langle \varepsilon, \alpha \rangle$: an action $\langle a, \varepsilon \rangle$ indicates that the orchestrator accepts a message a from the client, without delivering it to the service; dually, $\langle \overline{a}, \varepsilon \rangle$ indicates that the orchestrator sends a message $\overline{a}$ (previously received from the service) to the client; an action of the form $\langle \varepsilon, \alpha \rangle$ indicates a similar capability on the service side. *Synchronous actions* have the form $\langle a, \overline{a} \rangle$: they indicate that the orchestrator accepts a message a from the client, provided that the service can receive $\overline{a}$; dually for $\langle \overline{a}, a \rangle$. The orchestrator $f = \langle a, \overline{a} \rangle$

makes the client ρ' above compliant with $a+b$, because it disallows any interaction on b; the orchestrator $g = \langle c, \varepsilon\rangle.\langle a, \varepsilon\rangle.\langle \varepsilon, \overline{a}\rangle.\langle \varepsilon, \overline{c}\rangle.(\langle \overline{b}, b\rangle + \langle \overline{d}, d\rangle)$ makes the client ρ'' above compliant with σ, because the orchestrator accepts c followed by a from the client, and then delivers them in the order expected by the service. Orchestrators can be interpreted as morphisms transforming service contracts: the relation $f : a \preceq a + b$ states that every client satisfied by a is also satisfied by $a + b$ by means of the orchestrator f; the relation $g : c.a.(\overline{b} \oplus \overline{d}) \preceq a.c.(\overline{b} \oplus \overline{d})$ states that every client that sends $\overline{c}$ before $\overline{a}$ and then waits for either $\overline{b}$ or $\overline{d}$ can also be satisfied by $a.c.(\overline{b} \oplus \overline{d})$, provided that g orchestrates its interaction with the service. On the other hand, no orchestrator is able to make ρ interact successfully with σ', because the internal decisions taken by σ' cannot be controlled.

The reminder of the paper is structured as follows. In §2 we define syntax and semantics of the contract language and we define strong variants of the compliance and subcontract relations. These relations, which correspond directly to analogous notions in the standard testing framework, are too strict for the purposes of Web service discovery, as we have informally argued above. In §3 we define weak variants of compliance and subcontract relation, corresponding to the scenario where client and services interact while being mediated by a simple orchestrator. We proceed by studying simple orchestrators and the fundamental properties of the weak relations they induce, including their connection with the corresponding strong variants. §4 shows how to compute the dual contract in the presence of simple orchestrators and in §5 we provide an algorithm for synthesizing orchestrators by comparing service contracts. In §6 we show the algorithm at work on two less trivial examples and in §7 we informally overview some interesting subclasses of simple orchestrators that allow for efficient and particularly simple implementations. We conclude with an in-depth discussion of related work (§8) and a summary of the main contributions of this work (§9).

2 Contracts

The syntax of contracts makes use of a denumerable set $\mathscr{N}$ of *names* ranged over by $a, b, \ldots$ and of a denumerable set of *variables* ranged over by $x, y, \ldots$; we write $\overline{\mathscr{N}}$ for the set of *co-names* $\overline{a}$, where $a \in \mathscr{N}$. Names represent input actions, while co-names represent output actions; we let $\alpha, \beta, \ldots$ range over actions; we let $\varphi, \varphi', \ldots$ range over strings of actions, ε being the empty string as usual; we let $\mathrm{R}, \mathrm{S}, \ldots$ range over finite sets of actions; we let $\overline{\overline{\alpha}} = \alpha$ and $\overline{\mathrm{R}} = \{\overline{\alpha} \mid \alpha \in \mathrm{R}\}$ and $\overline{\varphi}$ be the sequence obtained by changing every action α in φ with its corresponding co-action $\overline{\alpha}$. The meaning of names is left unspecified: they can stand for ports, operations, message types, and so forth. Contracts are

ranged over by $\rho, \sigma, \tau, \ldots$ and their syntax is given by the following grammar:

$$\sigma ::= \mathtt{0} \mid \alpha.\sigma \mid \sigma + \sigma \mid \sigma \oplus \sigma \mid \mathtt{rec}\ x.\sigma \mid x$$

The notions of free and bound variables in contracts are standard, being $\mathtt{rec}\ x$ the only binder. In the following we write $\sigma\{^{\tau}/_{x}\}$ for the contract that is the same as σ except that every free occurrence of x has been replaced by τ. We assume variables to be *guarded*: every free occurrence of x in a term $\mathtt{rec}\ x.\sigma$ must be found in a subterm of σ having the form $\alpha.\sigma'$. The null contract $\mathtt{0}$ describes the idle process that offers no action (we will omit trailing $\mathtt{0}$'s); the contract $\alpha.\sigma$ describes a process that offers the action α and then behaves as σ; the contract $\sigma + \tau$ is the *external choice* of σ and τ and describes a process that can either behave as σ or as τ depending on the party it is interacting with; the contract $\sigma \oplus \tau$ is the *internal choice* of σ and τ and describes a process that autonomously decides to behave as either σ or τ; the contract $\mathtt{rec}\ x.\sigma$ describes a process that behaves as $\sigma\{^{\mathtt{rec}\ x.\sigma}/_{x}\}$.

Overall contracts are finite representations of possibly infinite regular trees generated by the $\mathtt{0}$, the prefix and the choice operators [16]. Recall that a regular tree always contains a *finite* number of different subtrees. The regular tree denoted by a contract is intuitively obtained by repeatedly unfolding every subterm $\mathtt{rec}\ x.\sigma$ to $\sigma\{^{\mathtt{rec}\ x.\sigma}/_{x}\}$. The guardedness condition stated above ensures that every infinite branch of the possibly infinite tree obtained by unfolding a contract contains infinite occurrences of the prefix operator. It excludes terms of the form $\mathtt{rec}\ x.x$ or $\mathtt{rec}\ x.x + x$ or $\mathtt{rec}\ x.x \oplus x$. All these terms usually represent *diverging* processes that we exclude from this work (the interested reader may refer to [31] for a treatment of divergence in the context of Web services).

The transition relation of contracts is inductively defined by the following rules (symmetric rules for $+$ and $\oplus$ are omitted):

$$\alpha.\sigma \xrightarrow{\alpha} \sigma \qquad \sigma \oplus \tau \longrightarrow \sigma \qquad \mathtt{rec}\ x.\sigma \longrightarrow \sigma\{^{\mathtt{rec}\ x.\sigma}/_{x}\}$$

$$\frac{\sigma \longrightarrow \sigma'}{\sigma + \tau \longrightarrow \sigma' + \tau} \qquad \frac{\sigma \xrightarrow{\alpha} \sigma'}{\sigma + \tau \xrightarrow{\alpha} \sigma'}$$

The relation $\longrightarrow$ denotes internal, invisible transitions, while $\xrightarrow{\alpha}$ denotes visible transitions labeled with an action α. The transition relation is the same as that of CCS without τ's [18]. In particular, the fact that $+$ stands for an external choice is clear from the fourth rule, where the internal transition $\sigma \longrightarrow \sigma'$ does not preempt the τ branch. The guardedness assumption we made earlier ensures that the number of consecutive internal transitions in any derivation of a contract is finite (*strong convergence*). We write $\Longrightarrow$ for the reflexive, transitive closure of $\longrightarrow$; let $\overset{\alpha}{\Longrightarrow}$ be $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$; we write $\sigma \xrightarrow{\alpha}$

if there exists σ' such that $\sigma \xrightarrow{\alpha} \sigma'$, and similarly for $\sigma \stackrel{\alpha}{\Longrightarrow}$; let $\mathtt{init}(\sigma) \stackrel{\text{def}}{=} \{\alpha \mid \sigma \stackrel{\alpha}{\Longrightarrow}\}$.

Definition 2.1 (strong compliance)**.** A *system* is a pair $\rho \parallel \sigma$ of a (client) contract ρ and a (service) contract σ interacting with each other. Let $\longrightarrow$ be the least relation between systems inductively defined as follows:

$$\frac{\rho \longrightarrow \rho'}{\rho \parallel \sigma \longrightarrow \rho' \parallel \sigma} \qquad \frac{\sigma \longrightarrow \sigma'}{\rho \parallel \sigma \longrightarrow \rho \parallel \sigma'} \qquad \frac{\rho \xrightarrow{\overline{\alpha}} \rho' \qquad \sigma \xrightarrow{\alpha} \sigma'}{\rho \parallel \sigma \longrightarrow \rho' \parallel \sigma'}$$

We write $\Longrightarrow$ for the reflexive, transitive closure of $\longrightarrow$; we write $\rho \parallel \sigma \not\longrightarrow$ if there exist no ρ' and σ' such that $\rho \parallel \sigma \longrightarrow \rho' \parallel \sigma'$. We say that ρ is *strongly compliant with* σ, notation $\rho \dashv \sigma$, if $\rho \parallel \sigma \Longrightarrow \rho' \parallel \sigma' \not\longrightarrow$ implies $\rho \xrightarrow{\mathsf{e}}$.

The first two rules in the definition of $\longrightarrow$ for systems indicate that client and service may evolve independently of each other by means of internal moves. The last rule describes a synchronization between client and service performing complementary actions. A client ρ is strongly compliant with a service σ if every computation of the system $\rho \parallel \sigma$ reaching a stable state $\rho' \parallel \sigma'$ is such that $\rho' \xrightarrow{\mathsf{e}}$, which denotes the successful termination of the client. For instance $a.\mathsf{e} + b.\mathsf{e} \dashv \overline{a} \oplus \overline{b}$ and $a.\mathsf{e} \oplus b.\mathsf{e} \dashv \overline{a} + \overline{b}$, but $a.\mathsf{e} \oplus b.\mathsf{e} \not\dashv \overline{a} \oplus \overline{b}$ because of the computation $a.\mathsf{e} \oplus b.\mathsf{e} \parallel \overline{a} \oplus \overline{b} \Longrightarrow a.\mathsf{e} \parallel \overline{b} \not\longrightarrow$.

The (strong) compliance relation provides us with the most natural equivalence for comparing services: the (service) contract σ is "smaller than" the (service) contract τ if every client that is compliant with σ is also compliant with τ.

Definition 2.2 (strong subcontract)**.** We say that σ is a *strong subcontract* of τ, notation $\sigma \sqsubseteq \tau$, if $\rho \dashv \sigma$ implies $\rho \dashv \tau$ for every ρ. We write $\simeq$ for the equivalence relation induced by $\sqsubseteq$, that is $\simeq \,= \sqsubseteq \cap \sqsupseteq$.

For instance, we have $a \oplus b \sqsubseteq a$ because every client that is satisfied by a service that may decide to offer either a or b is also satisfied by a service that systematically offers a. On the other hand $a.(\overline{b} + \overline{d}) \not\sqsubseteq a.\overline{b} + a.\overline{d}$ since $\overline{a}.b.\mathsf{e} \dashv a.(\overline{b} + \overline{d})$ but $\overline{a}.b.\mathsf{e} \not\dashv a.\overline{b} + a.\overline{d}$ because of the computation $\overline{a}.b.\mathsf{e} \parallel a.\overline{b} + a.\overline{d} \longrightarrow b.\mathsf{e} \parallel \overline{d} \not\longrightarrow$. In the last example a client of $a.(\overline{b} + \overline{d})$ can decide whether to receive $\overline{b}$ or $\overline{d}$ after sending $\overline{a}$, whereas in $a.\overline{b} + a.\overline{d}$ only one of these actions is available, according to the branch taken by the service. In fact it is possible to prove that $a.\overline{b} + a.\overline{d} \simeq a.(\overline{b} \oplus \overline{d})$.

The set-theoretic definition of the preorder above admits another, more intuitive characterization which is also propedeutic to the alternative characterization of the weak subcontract relation in §3. In order to define it we need two auxiliary notions, that of contract continuation and of ready set.

The transition relation of contracts describes the evolution of a contract from the point of view of the process exposing, or implementing, the contract. The notion of *contract continuation*, which we are to define next, considers the point of view of the process it is interacting with.

Definition 2.3 (contract continuation)**.** Let $\sigma \stackrel{\alpha}{\Longrightarrow}$. The *continuation* of σ with respect to α, notation $\sigma(\alpha)$, is defined as $\sigma(\alpha) \stackrel{\text{def}}{=} \bigoplus_{\sigma \Longrightarrow \stackrel{\alpha}{\longrightarrow} \sigma'} \sigma'$. We generalize the notion of continuation to finite sequences of actions so that $\sigma(\varepsilon) = \sigma$ and $\sigma(\alpha\varphi) = \sigma(\alpha)(\varphi)$.

For example, $a.\overline{b} + a.\overline{d} \stackrel{a}{\longrightarrow} \overline{b}$ (the process knows which branch has been taken after an action a) but $(a.\overline{b}+a.\overline{d})(a) = \overline{b} \oplus \overline{d}$ (the party interacting with $a.\overline{b}+a.\overline{d}$ does not know which branch has been taken after seeing an a action, hence it considers both). Because of the guardedness condition there is a finite number of residuals σ' such that $\sigma \Longrightarrow \stackrel{\alpha}{\longrightarrow} \sigma'$, hence $\sigma(\alpha)$ is well defined. We can state an even stronger property which we will implicitly use throughout the paper for asserting the well-foundedness of several definitions.

Proposition 2.4. *Let* $D(\sigma) \stackrel{\text{def}}{=} \{\sigma(\varphi) \mid \sigma \stackrel{\varphi}{\Longrightarrow}\}$. *Then* $D(\sigma)$ *is finite for every* σ.

Proof. It is sufficient to observe that $\sigma(\varphi)$ is the internal choice of some subtrees in the regular tree resulting from the unfolding of σ. Since a regular tree has finite different subtrees, there is a finite number of terms $\sigma(\varphi)$. An alternative, direct proof can be found in [32]. □

The *ready sets* of a contract tell us about its internal nondeterminism.

Definition 2.5 (ready set)**.** We say that σ has *ready set* R, written $\sigma \Downarrow \mathrm{R}$, if $\sigma \Longrightarrow \sigma' \not\longrightarrow$ and $\mathrm{R} = \mathtt{init}(\sigma')$.

Intuitively, $\sigma \Downarrow \mathrm{R}$ means that σ can independently evolve, by means of internal transitions, to a stable contract σ' which only offers the actions in R. For example, $\{a, b\}$ is the only ready set of $a+b$ (both a and b are always available), whereas the ready sets of $a \oplus b$ are $\{a\}$ and $\{b\}$ (the contract $a \oplus b$ may evolve into a state where only a is available, or into a state where only b is available). Similarly, $a + (b \oplus c)$ has two ready sets $\{a, b\}$ and $\{a, c\}$. Namely, the availability of action a is always guaranteed (it can be chosen externally, see "+" in the contract), but only one of b or c will be available (the choice of which is made internally, see "$\oplus$" in the contract).

We are now ready to define an alternative characterization of $\sqsubseteq$.

Definition 2.6 (coinductive strong subcontract)**.** We say that $\mathscr{S}$ is a *coinductive strong subcontract relation* if $(\sigma, \tau) \in \mathscr{S}$ implies

(1) $\tau \Downarrow \mathrm{S}$ implies $\sigma \Downarrow \mathrm{R}$ and $\mathrm{R} \subseteq \mathrm{S}$ for some R, and
(2) $\tau \stackrel{\alpha}{\Longrightarrow}$ implies $\sigma \stackrel{\alpha}{\Longrightarrow}$ and $(\sigma(\alpha), \tau(\alpha)) \in \mathscr{S}$.

Condition (1) requires τ to be more deterministic than σ (every ready set of τ has a corresponding one of σ that offers fewer actions). Condition (2) requires τ to offer no more actions than those offered by σ, and every continuation after an action offered by both σ and τ to be in the subcontract relation. We conclude this section with a summary of the most important properties enjoyed by $\sqsubseteq$.

Proposition 2.7. *The following properties hold:*

(1) $\sqsubseteq$ is the largest coinductive subcontract relation;
(2) $\sqsubseteq$ coincides with the must *preorder [18, 17, 26] for strongly convergent processes;*
(3) $\sqsubseteq$ is a precongruence with respect to all the operators of the contract language.

Proof. We only prove item (1). A proof of item (2) can be found in [31] and precongruence proofs for the must preorder can be found in [26].

First of all we prove that $\sqsubseteq$ is a coinductive strong subcontract relation. Let $\sigma \sqsubseteq \tau$. As regards condition (1) in Definition 2.6, let $\{\mathrm{R}_1, \ldots, \mathrm{R}_n\}$ be the ready sets of σ and assume by contradiction that there exists S such that $\tau \Downarrow \mathrm{S}$ and $\mathrm{R}_i \not\subseteq \mathrm{S}$ for every $1 \leq i \leq n$. Namely, for every $1 \leq i \leq n$ there exists $\alpha_i \in \mathrm{R}_i \setminus \mathrm{S}$. Consider $\rho \stackrel{\text{def}}{=} \sum_{1 \leq i \leq n} \overline{\alpha}_i.\mathtt{e}$. Then $\rho \dashv \sigma$ but $\rho \not\dashv \tau$, which is absurd by hypothesis. As regards condition (2), let $\tau \stackrel{\alpha}{\Longrightarrow}$ and assume by contradiction that $\sigma \stackrel{\alpha}{\not\Longrightarrow}$. Consider $\rho \stackrel{\text{def}}{=} \mathtt{e} + \overline{\alpha}$. Then $\rho \dashv \sigma$ but $\rho \not\dashv \tau$, which is absurd by hypothesis. Let ρ' be a client contract such that $\rho' \dashv \sigma(\alpha)$ and consider $\rho \stackrel{\text{def}}{=} \mathtt{e} + \overline{\alpha}.\rho'$. Then $\rho \dashv \sigma$, from which we derive $\rho \dashv \tau$, hence $\rho' \dashv \tau(\alpha)$. We conclude $\sigma(\alpha) \sqsubseteq \tau(\alpha)$ because ρ' is arbitrary.

Then we prove that $\sqsubseteq$ is the largest among all the coinductive strong subcontract relations. To this aim it is sufficient to show that any coinductive strong subcontract relation $\mathscr{S}$ is included in $\sqsubseteq$. Let $(\sigma, \tau) \in \mathscr{S}$ and assume $\rho \dashv \sigma$. Consider now a maximal computation $\rho \parallel \tau \Longrightarrow \rho' \parallel \tau' \not\longrightarrow$. We can "unzip" this derivation into two derivations $\rho \stackrel{\overline{\varphi}}{\Longrightarrow} \rho' \not\longrightarrow$ and $\tau \stackrel{\varphi}{\Longrightarrow} \tau' \not\longrightarrow$ for some string φ of actions. From condition (2) of Definition 2.6 and by induction on φ we derive that $\sigma \stackrel{\varphi}{\Longrightarrow}$ and $(\sigma(\varphi), \tau(\varphi)) \in \mathscr{S}$. From $\tau(\varphi) \Downarrow \mathtt{init}(\tau')$ and condition (1) of Definition 2.6 we derive that there exists $\mathrm{R} \subseteq \mathtt{init}(\tau')$ such that $\sigma(\varphi) \Downarrow \mathrm{R}$. By definition of ready set we obtain that there exists σ' such that $\sigma \stackrel{\varphi}{\Longrightarrow} \sigma' \not\longrightarrow$ and $\mathtt{init}(\sigma') \subseteq \mathtt{init}(\tau')$. We can now "zip" the two derivations starting from ρ and σ and obtain a derivation $\rho \parallel \sigma \Longrightarrow \rho' \parallel \sigma'$. We observe that $\rho' \parallel \sigma' \not\longrightarrow$ since $\rho' \not\longrightarrow$ and $\sigma' \not\longrightarrow$ and $\mathtt{init}(\sigma') \subseteq \mathtt{init}(\tau')$. From $\rho \dashv \sigma$

we conclude $\rho' \xrightarrow{\mathsf{e}}$. $\square$

Property (1) states the correctness of Definition 2.6 as an alternative characterization for $\sqsubseteq$. Property (2) connects $\sqsubseteq$ with the well-known *must* testing preorder. This result is not entirely obvious because the notion of "passing a test" we use differs from that used in the standard testing framework (see [31] for more details). Finally, property (3) states that $\sqsubseteq$ is well behaved and that it can be used for modular refinement. The weak variant of the subcontract relation that we will define in §3 does not enjoy this property in general, but not without reason as we will see.

3 Simple orchestrators

The strong compliance relation (Definition 2.1) is based on interactions where at each synchronization progress is always guaranteed for *both* client and service. We relax this requirement and assume that an orchestrator mediates the interaction of a client and a service. The orchestrator ensures that at each synchronization progress is guaranteed for *at least one* of the interacting parties. The orchestrator must be *fair*, in the sense that client and service must have equal opportunities to make progress. In other words, the orchestrator should not indefinitely guarantee progress to only one of the two parties. Also, the orchestrator must not disrupt the communication flow between client and service: it cannot bounce a message back to the same party that sent it.

3.1 Weak compliance and subcontract relations

Orchestration actions are described by the following grammar:

$$\mu ::= \langle \alpha, \varepsilon \rangle \mid \langle \varepsilon, \alpha \rangle \mid \langle a, \overline{a} \rangle \mid \langle \overline{a}, a \rangle$$

The action $\langle \alpha, \varepsilon \rangle$ means that the orchestrator offers α to the client; the action $\langle \varepsilon, \alpha \rangle$ means that the orchestrator offers α to the service; the action $\langle \alpha, \overline{\alpha} \rangle$ means that the orchestrator simultaneously offers α to the client and $\overline{\alpha}$ to the service. Actions $\langle \alpha, \varepsilon \rangle$ and $\langle \varepsilon, \alpha \rangle$ are called *asynchronous* orchestration actions because, if executed, they guarantee progress to only one party among client and service. On the other hand, $\langle \alpha, \overline{\alpha} \rangle$ are *synchronous* orchestration actions because, if executed, they guarantee simultaneous progress to both client and service. We let $\mu, \mu', \ldots$ range over orchestration actions and $\mathrm{A}, \mathrm{A}', \ldots$ range over sets of orchestrator actions.

A *directional buffer* is a map $\{\circ, \bullet\} \times \overline{\mathscr{N}} \to \mathbb{Z}$ associating pairs $(r, \overline{a})$ with the

number of $\overline{a}$ messages stored in the buffer and available for delivery to the role r, where r can be $\circ$ for "client" or $\bullet$ for "service"; we let $\mathbb{B}, \mathbb{B}', \ldots$ range over buffers. Directionality is ensured by distinguishing messages to be delivered to the client from messages to be delivered to the service. For technical reasons we allow $\mathtt{cod}(\mathbb{B})$ – the codomain of $\mathbb{B}$ – to range over $\mathbb{Z}$, although every well-formed buffer will always contain a nonnegative number of messages. We write $\tilde{\emptyset}$ for the empty buffer, the one having $\{0\}$ as codomain; we write $\mathbb{B}[(r, \overline{a}) \mapsto n]$ for the buffer $\mathbb{B}'$ which is the same as $\mathbb{B}$ except that $(r, \overline{a})$ is associated with n; we write $\mathbb{B}\mu$ for the buffer $\mathbb{B}$ updated after the orchestration action μ:

$$\begin{array}{rll}
\mathbb{B}\langle a, \varepsilon\rangle &= \mathbb{B}[(\bullet, \overline{a}) \mapsto \mathbb{B}(\bullet, \overline{a}) + 1] & \text{(accept } \overline{a} \text{ from the client)} \\
\mathbb{B}\langle \overline{a}, \varepsilon\rangle &= \mathbb{B}[(\circ, \overline{a}) \mapsto \mathbb{B}(\circ, \overline{a}) - 1] & \text{(send } \overline{a} \text{ to the client)} \\
\mathbb{B}\langle \varepsilon, a\rangle &= \mathbb{B}[(\circ, \overline{a}) \mapsto \mathbb{B}(\circ, \overline{a}) + 1] & \text{(accept } \overline{a} \text{ from the service)} \\
\mathbb{B}\langle \varepsilon, \overline{a}\rangle &= \mathbb{B}[(\bullet, \overline{a}) \mapsto \mathbb{B}(\bullet, \overline{a}) - 1] & \text{(send } \overline{a} \text{ to the service)} \\
\mathbb{B}\langle \alpha, \overline{\alpha}\rangle &= \mathbb{B} & \text{(synchronize client and service)}
\end{array}$$

We say that $\mathbb{B}$ has rank k, or is a k-buffer, if $\mathtt{cod}(\mathbb{B}) \subseteq [0, k]$; we say that the k-buffer $\mathbb{B}$ *enables* the orchestration action μ, notation $\mathbb{B} \vdash_k \mu$, if $\mathbb{B}\mu$ is still a k-buffer. For instance $\tilde{\emptyset} \vdash_1 \langle a, \varepsilon\rangle$ but $\tilde{\emptyset} \nvdash_k \langle \overline{a}, \varepsilon\rangle$ because $-1 \in \mathtt{cod}(\tilde{\emptyset}\langle \overline{a}, \varepsilon\rangle)$. We extend the notion to sets of actions so that $\mathbb{B} \vdash_k \mathrm{A}$ if $\mathbb{B}$ enables every action in A. Synchronization actions are enabled regardless of the rank of the buffer, because they leave the buffer unchanged.

The language of simple orchestrators is defined by the following grammar:

$$f ::= \mathtt{0} \mid \mu.f \mid f \vee f \mid f \wedge f \mid \mathtt{rec}\ x.f \mid x$$

We let $f, g, h, \ldots$ range over orchestrators. The orchestrator $\mathtt{0}$ offers no action (we will omit trailing $\mathtt{0}$'s); the orchestrator $\mu.f$ offers the action μ and then continues as f; the orchestrator $f \vee g$ offers the actions offered by either f or g; the orchestrator $f \wedge g$ offers the actions offered by both f and g; recursive orchestrators can be expressed by means of $\mathtt{rec}\ x.f$ and recursion variables in the usual way. As for contracts, we make the assumption that recursion variables must be guarded by at least one orchestration action.

Simple orchestrators do not exhibit internal nondeterminism. This calls for an operational semantics merely expressing which orchestration actions are available (symmetric rule for $\vee$ omitted):

$$\mu.f \xrightarrow{\mu} f \qquad \frac{f \xrightarrow{\mu} f'}{f \vee g \xrightarrow{\mu} f'} \qquad \frac{f \xrightarrow{\mu} f' \quad g \xrightarrow{\mu} g'}{f \wedge g \xrightarrow{\mu} f' \wedge g'} \qquad \frac{f\{\mathtt{rec}\ x.f/_x\} \xrightarrow{\mu} f'}{\mathtt{rec}\ x.f \xrightarrow{\mu} f'}$$

In practice we will identify orchestrators with the set $[\![f]\!]$ of strings of orchestration actions they offer, namely

$$[\![f]\!] \stackrel{\text{def}}{=} \{\mu_1 \cdots \mu_n \mid \exists g : f \xrightarrow{\mu_1} \cdots \xrightarrow{\mu_n} g\}$$

Observe that $[\![f]\!]$ is a non-empty, prefix-closed, regular set of strings of orchestration actions. We write $f \stackrel{\mu}{\longmapsto} g$ if $[\![g]\!] = \{\mu_1 \cdots \mu_n \mid \mu\mu_1 \cdots \mu_n \in [\![f]\!]\}$, namely g is the residual of f after the action μ. Note that $f \stackrel{\mu}{\longmapsto} f'$ and $f \stackrel{\mu}{\longmapsto} f''$ implies $[\![f']\!] = [\![f'']\!]$. We write $f \stackrel{\mu_1 \cdots \mu_n}{\longmapsto}$ if $\mu_1 \cdots \mu_n \in [\![f]\!]$; we write $f \stackrel{\mu}{\not\longmapsto}$ if $\mu \notin [\![f]\!]$; let $\mathtt{init}(f) \stackrel{\text{def}}{=} \{\mu \mid f \stackrel{\mu}{\longmapsto}\}$. We say that f is a *valid orchestrator of rank k*, or is a k-orchestrator, if $f \stackrel{\mu_1 \cdots \mu_n}{\longmapsto}$ implies that $\tilde{\emptyset}\mu_1 \cdots \mu_n$ is a k-buffer. Not every term f denotes a valid orchestrator of finite rank. For instance $\mathtt{rec}\ x.\langle a, \varepsilon\rangle.x$ is invalid because it accepts an unbounded number of messages from the client; $\langle \overline{a}, \varepsilon\rangle$ is invalid because it tries to deliver a message that it has not received; $\langle \varepsilon, a\rangle.\langle \overline{a}, \varepsilon\rangle$ is a valid orchestrator of rank 1 (or greater). In the following we will always work with valid orchestrators of finite rank.

A better intuition of the semantics of orchestrator can be given by inspecting directly the weak variant of the compliance relation, where client and service interact under the supervision of an orchestrator.

Definition 3.1 (weak compliance relation)**.** An *orchestrated system* is a triple $\rho \parallel_f \sigma$ of a (client) contract ρ and a (service) contract σ interacting with each other while being supervised by an orchestrator f. Let $\longrightarrow$ be the least relation between orchestrated systems inductively defined as follows:

$$\frac{\rho \longrightarrow \rho'}{\rho \parallel_f \sigma \longrightarrow \rho' \parallel_f \sigma} \qquad \frac{\sigma \longrightarrow \sigma'}{\rho \parallel_f \sigma \longrightarrow \rho \parallel_f \sigma'} \qquad \frac{\rho \stackrel{\overline{\alpha}}{\longrightarrow} \rho' \quad f \stackrel{\langle \alpha, \overline{\alpha}\rangle}{\longmapsto} f' \quad \sigma \stackrel{\alpha}{\longrightarrow} \sigma'}{\rho \parallel_f \sigma \longrightarrow \rho' \parallel_{f'} \sigma'}$$

$$\frac{\rho \stackrel{\overline{\alpha}}{\longrightarrow} \rho' \quad f \stackrel{\langle \alpha, \varepsilon\rangle}{\longmapsto} f'}{\rho \parallel_f \sigma \longrightarrow \rho' \parallel_{f'} \sigma} \qquad \frac{f \stackrel{\langle \varepsilon, \overline{\alpha}\rangle}{\longmapsto} f' \quad \sigma \stackrel{\alpha}{\longrightarrow} \sigma'}{\rho \parallel_f \sigma \longrightarrow \rho \parallel_{f'} \sigma'}$$

We write $\Longrightarrow$ for the reflexive, transitive closure of $\longrightarrow$; we write $\rho \parallel_f \sigma \not\longrightarrow$ if there exist no ρ', f', and σ' such that $\rho \parallel_f \sigma \longrightarrow \rho' \parallel_{f'} \sigma'$. We write $f : \rho \dashv\!\mid \sigma$ if $\rho \parallel_f \sigma \Longrightarrow \rho' \parallel_{f'} \sigma' \not\longrightarrow$ implies $\rho' \stackrel{\mathsf{e}}{\longrightarrow}$. We say that ρ is *weakly k-compliant with σ*, notation $\rho \dashv\!\mid_k \sigma$, if there exists a k-orchestrator f such that $f : \rho \dashv\!\mid \sigma$.

The first two rules in the definition of $\longrightarrow$ for orchestrated systems are basically the same as for the strong compliance relation. In particular the orchestrator has no way to affect the internal moves of client and service. The third rule expresses the fact that client and service can synchronize with each other, but only if the orchestrator permits it (the action $\langle \alpha, \overline{\alpha}\rangle$ "connects" the action $\overline{\alpha}$ performed by the client with the action α performed by the service). The last two rules express the fact that client and service may interact with the orchestrator, independently of the other partner, if the orchestrator provides suitable asynchronous actions. Observe that in each rule progress is guaranteed for at least one of the interacting parties.

As an example of weak compliance we have $\overline{a}.\mathsf{e} + \overline{b}.d.\mathsf{e} \dashv\!\!\mid_0 a + b.\overline{c}$ because the orchestrator $\langle a, \overline{a}\rangle$ disallows the interaction on b at the first step while permitting the interaction on a. On the other hand $\overline{a}.\mathsf{e} \oplus \overline{b}.d.\mathsf{e} \not\dashv\!\!\mid_0 a + b.\overline{c}$ because no orchestrator can prevent the client from autonomously evolving to $\overline{b}.d.\mathsf{e}$. At this point, even if the synchronization on b is possible, client and service will get stuck at the next step. As another example, we have $\overline{a}.\overline{c}.b.\mathsf{e} \dashv\!\!\mid_1 c.a.\overline{b}$, by means of the orchestrator $\langle a, \varepsilon\rangle.\langle c, \varepsilon\rangle.\langle \varepsilon, \overline{c}\rangle.\langle \varepsilon, \overline{a}\rangle.\langle \overline{b}, b\rangle$ which accepts the messages in the order required by the client, but delivers them in the order expected by the service.

Weak compliance induces the weak subcontract relation as follows:

Definition 3.2 (weak subcontract)**.** We say that σ is a *weak k-subcontract* of τ, notation $\sigma \preceq_k \tau$, if $\rho \dashv \sigma$ implies $\rho \dashv\!\!\mid_k \tau$ for every ρ.

Namely, $\sigma \preceq_k \tau$ implies that a service with contract τ can replace a service with contract σ because every client satisfied by σ (that is $\rho \dashv \sigma$) can also be satisfied by τ (that is $\rho \dashv\!\!\mid_k \tau$) by means of some k-orchestrator f. In the following we will drop the index k from $\dashv\!\!\mid_k$ and $\preceq_k$ when immaterial.

Whether or not $\preceq$ is the subcontract relation we are looking for is hard to tell from Definition 3.2. In part this is because it is very reasonable to expect that the k-orchestrator f may depend on the particular client ρ that we are considering. In addition, it is not even obvious that $\preceq$ is transitive, which is required if we plan to use $\preceq$ as stated in the introduction. We will thus devote the following subsections to the study of $\preceq$ and of its main properties.

3.2 Basic properties of the weak subcontract relation

Among all the orchestrators involved in a relation $\sigma \preceq \tau$, we can restrict our interest to a relatively small class of *relevant* ones.

Definition 3.3 (relevant orchestrator)**.** Let $\sigma \preceq_k \tau$ and f be a k-orchestrator such that $f : \rho \dashv\!\!\mid \tau$. We say that f is *relevant* for $\sigma \preceq_k \tau$ if $\sigma \xRightarrow{\varphi_1 \cdots \varphi_n}$ and $f \xrightarrow{\langle \varphi_1, \overline{\varphi}'_1\rangle \cdots \langle \varphi_n, \overline{\varphi}'_n\rangle \langle \varphi, \overline{\varphi}'\rangle}$ and $\tau \xRightarrow{\varphi'_1 \cdots \varphi'_n}$ imply $\sigma(\varphi_1 \cdots \varphi_n) \Longrightarrow$ and $\tau(\varphi'_1 \cdots \varphi'_n) \Longrightarrow$.

An orchestrator that is relevant for $\sigma \preceq \tau$ never offers orchestration actions that do not correspond to actions offered by σ and that would never be enabled by τ. The fact that it is always possible to find a relevant orchestrator that is able to satisfy a particular client is a first step towards a theory of contracts that is client-independent. The proof of this fact is not entirely obvious. It is clear that actions of the form $\langle \alpha, \overline{\alpha}\rangle$ and $\langle \varepsilon, \overline{\alpha}\rangle$ can be safely removed if τ does not offer corresponding co-actions. However, asynchronous actions of the form $\langle \alpha, \varepsilon\rangle$ may actually be necessary for the orchestrator to satisfy the client, even

if σ never offers α actions. For instance, $\overline{a}.\mathtt{e} + \overline{b}.\mathtt{e} + \overline{c}.\overline{a}.\mathtt{e} \dashv a \oplus b$ and $a \oplus b \preceq a$ and $\langle c, \varepsilon\rangle.\langle a, \overline{a}\rangle : \overline{a}.\mathtt{e} + \overline{b}.\mathtt{e} + \overline{c}.\overline{a}.\mathtt{e} \dashv\!\shortmid a$. Simply removing the $\langle c, \varepsilon\rangle$ action (and the corresponding continuation) would produce the null orchestrator, which clearly cannot satisfy the client.

Proposition 3.4. *Let $\sigma \preceq_k \tau$ and $\rho \dashv \sigma$ and f be a k-orchestrator such that $f : \rho \dashv\!\shortmid \tau$. Then there exists a k-orchestrator g relevant for $\sigma \preceq_k \tau$ such that $g : \rho \dashv\!\shortmid \tau$.*

Proof. We say that a subterm $\overline{\alpha}.\rho'$ of ρ is useless if $\rho \stackrel{\overline{\varphi}}{\Longrightarrow} \xrightarrow{\overline{\alpha}} \rho'$ and $\sigma \stackrel{\varphi}{\Longrightarrow}$ implies $\sigma(\varphi) \stackrel{\varphi}{\Longrightarrow}$. Let ρ_r be the (client) contract obtained from ρ by replacing every useless subterm $\overline{\alpha}.\rho'$ with $\overline{\alpha}.\mathtt{0}$. Clearly $\rho_r \dashv \sigma$ since no synchronization will ever occur on those $\overline{\alpha}$ actions that guard useless subterms of ρ. From the hypothesis $\sigma \preceq_k \tau$ there exists a k-orchestrator g such that $g : \rho_r \dashv\!\shortmid \tau$. Let

$$R(g, \sigma, \tau) \stackrel{\text{def}}{=} \bigvee_{g \xrightarrow{\langle\varphi,\overline{\varphi}'\rangle} g', \sigma \stackrel{\varphi}{\Longrightarrow}, \tau \xrightarrow{\varphi'}} \langle \varphi, \overline{\varphi}'\rangle.R(g', \sigma(\varphi), \tau(\varphi'))$$

and let $g_r \stackrel{\text{def}}{=} R(g, \sigma, \tau)$. Observe that g_r is well defined because σ, τ, and g are regular. Observe also that g_r is relevant for $\sigma \preceq_k \tau$ by its own definition and $g_r : \rho_r \dashv\!\shortmid \tau$ because every derivation starting from $\rho_r \|_{g_r} \tau$ is also a possible derivation starting from $\rho_r \|_g \tau$. We prove that $g_r : \rho \dashv\!\shortmid \tau$. Consider a derivation $\rho \|_{g_r} \tau \Longrightarrow \rho' \|_{g'_r} \tau' \not\longrightarrow$. Then there exist $\varphi_1, \ldots, \varphi_n$ and $\varphi'_1, \ldots, \varphi'_n$ such that $\rho \xrightarrow{\overline{\varphi}_1 \cdots \overline{\varphi}_n} \rho' \not\longrightarrow$ and $g_r \xrightarrow{\langle\varphi_1,\overline{\varphi}'_1\rangle \cdots \langle\varphi_n,\overline{\varphi}'_n\rangle} g'_r$ and $\tau \xrightarrow{\varphi'_1 \cdots \varphi'_n} \tau' \not\longrightarrow$. None of the φ_i can be an α guarding a useless subterm of ρ, by construction of g_r. By definition of ρ_r, there exists ρ'_r such that $\rho_r \xrightarrow{\overline{\varphi}_1 \cdots \overline{\varphi}_n} \rho'_r$ and $\mathtt{init}(\rho'_r) = \mathtt{init}(\rho')$ (in fact it is possible to find a ρ'_r that is the same as ρ' except that useless subterms $\overline{\alpha}.\rho''$ have been replaced by $\overline{\alpha}.\mathtt{0}$). By zipping the derivations starting from ρ_r, g_r, and τ we obtain $\rho_r \|_{g_r} \tau \Longrightarrow \rho'_r \|_{g'_r} \tau'$ and we notice that $\rho'_r \|_{g'_r} \tau' \not\longrightarrow$ since $\mathtt{init}(\rho'_r) = \mathtt{init}(\rho')$. From $g_r : \rho_r \dashv\!\shortmid \tau$ we deduce $\rho'_r \xrightarrow{\mathtt{e}}$, hence we conclude $\rho' \xrightarrow{\mathtt{e}}$. □

The relation $\sigma \preceq \tau$ means that every client ρ satisfied by σ is weakly compliant with τ by means of *some* orchestrator which, in principle, may depend on ρ. The next result shows that it is always possible to find an orchestrator that makes τ work seamlessly with *every* client satisfied by σ. We call such orchestrator "universal", since it is independent of a particular client.

Definition 3.5 (universal orchestrator)**.** We say that f is a *universal orchestrator* proving $\sigma \preceq_k \tau$, notation $f : \sigma \preceq_k \tau$, if f is a k-orchestrator and $\rho \dashv \sigma$ implies $f : \rho \dashv\!\shortmid \tau$ for every ρ.

On the theoretical side, the existence of the universal orchestrator allows us to study the properties of $\preceq$ independently of specific clients. On the practical

side, it makes it possible to precompute not only the subcontract relation $\preceq$ but also the orchestrator that proves it, regardless of the client performing the query.

Proposition 3.6. *$\sigma \preceq_k \tau$ if and only if there exists a k-orchestrator f such that $\rho \dashv \sigma$ implies $f : \rho \dashv\!\shortmid \tau$ for every ρ.*

Proof. The "if" part is trivial. As regards the "only if" part, the intuition is that if we are able to find the "most demanding" client satisfied by σ, then its corresponding orchestrator is universal. The most demanding client satisfied by σ, denoted by $\sigma^\top$, can be defined thus:

$$\sigma^\top \stackrel{\text{def}}{=} \sum_{\sigma \Downarrow \mathrm{R}} \begin{cases} \mathsf{e} & \text{if } \mathrm{R} = \emptyset \\ \bigoplus_{\alpha \in \mathrm{R}} \overline{\alpha}.\sigma(\alpha)^\top & \text{otherwise} \end{cases}$$

Well-foundedness of this definition follows from the regularity of σ. It is trivial to verify that $\sigma^\top \dashv \sigma$. Let f be a k-orchestrator such that $f : \sigma^\top \dashv\!\shortmid \tau$ and assume, without loss of generality, that f is relevant for $\sigma \preceq_k \tau$. We prove that $f : \sigma \preceq_k \tau$ by contradiction. Assume that there exists ρ such that $\rho \dashv \sigma$ and $f : \rho \not\dashv\!\shortmid \tau$. Then there exists a derivation $\rho \| _f \tau \Longrightarrow \rho' \| _{f'} \tau' \not\longrightarrow$ such that $\rho' \stackrel{\mathsf{e}}{\not\longrightarrow}$. Consequently there exist $\varphi_1, \ldots, \varphi_n$ and $\varphi'_1, \ldots, \varphi'_n$ such that $\rho \stackrel{\overline{\varphi}_1 \cdots \overline{\varphi}_n}{\Longrightarrow} \rho' \not\longrightarrow$ and $\tau \stackrel{\varphi'_1 \cdots \varphi'_n}{\Longrightarrow} \tau' \not\longrightarrow$ and $f \stackrel{\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle}{\longmapsto} f'$. Since f is relevant, it is easy to deduce, by induction on n, that $\sigma \stackrel{\varphi_1 \cdots \varphi_n}{\Longrightarrow}$. From $\rho' \|_{f'} \tau' \not\longrightarrow$ we deduce that $\rho' \stackrel{\overline{\alpha}}{\longrightarrow}$ implies $f' \stackrel{\langle \alpha, \varepsilon \rangle}{\longmapsto}$ and $f' \stackrel{\langle \alpha, \overline{\alpha} \rangle}{\longmapsto}$ implies $\tau' \stackrel{\alpha}{\not\longrightarrow}$. Let $\mathrm{R}_1, \ldots, \mathrm{R}_m$ be the ready sets of $\sigma(\varphi_1 \cdots \varphi_n)$. From $\rho \dashv \sigma$ and $\rho' \stackrel{\mathsf{e}}{\not\longrightarrow}$ we deduce that for every $1 \leq i \leq n$ there exists $\alpha_i \in \mathrm{R}_i$ and $\rho' \stackrel{\overline{\alpha}_i}{\longrightarrow}$. By definition of most demanding client we have $\sigma(\varphi_1 \cdots \varphi_n)^\top \Downarrow \{\overline{\alpha}_1, \ldots, \overline{\alpha}_m\}$, because each ready set of $\sigma(\varphi_1 \cdots \varphi_n)^\top$ is obtained by taking one action from every non-empty ready set of $\sigma(\varphi_1 \cdots \varphi_n)$. Hence there exists ρ'' such that $\sigma^\top \stackrel{\overline{\varphi}_1 \cdots \overline{\varphi}_n}{\Longrightarrow} \rho'' \not\longrightarrow$ and $\mathtt{init}(\rho'') = \{\overline{\alpha}_1, \ldots, \overline{\alpha}_m\} \subseteq \mathtt{init}(\rho')$. By zipping the derivations starting from $\sigma^\top$, f, and τ we obtain a derivation $\sigma^\top \|_f \tau \Longrightarrow \rho'' \|_{f'} \tau' \not\longrightarrow$ where $\rho'' \stackrel{\mathsf{e}}{\not\longrightarrow}$. This is absurd, for $f : \sigma^\top \dashv\!\shortmid \tau$ by hypothesis. □

3.3 Coinductive characterization of weak subcontract

In order to provide an alternative characterization of $\preceq$, which will also guide us in defining the algorithm for deciding $\preceq$ in §5, we need to know the effect of orchestration actions on the ready sets of client and service as perceived by the corresponding partner. When an orchestrator mediates the interaction between a client and a service, it proposes at each interaction step a set of orchestration actions A. If R is a client ready set and S is a service ready set,

then $\mathrm{A} \circ \mathrm{S}$ denotes the service ready set perceived by the client and $\mathrm{R} \bullet \mathrm{A}$ denotes the client ready set perceived by the service:

$$\mathrm{A} \circ \mathrm{S} \stackrel{\text{def}}{=} \{\alpha \mid \langle \alpha, \varepsilon \rangle \in \mathrm{A}\} \cup \{\alpha \in \mathrm{S} \mid \langle \alpha, \overline{\alpha} \rangle \in \mathrm{A}\}$$
$$\mathrm{R} \bullet \mathrm{A} \stackrel{\text{def}}{=} \{\overline{\alpha} \mid \langle \varepsilon, \overline{\alpha} \rangle \in \mathrm{A}\} \cup \{\overline{\alpha} \in \mathrm{R} \mid \langle \alpha, \overline{\alpha} \rangle \in \mathrm{A}\}$$

Namely, the client sees an action α if either that action is provided asynchronously by the orchestrator ($\langle \alpha, \varepsilon \rangle \in \mathrm{A}$), or if it is provided by the service ($\alpha \in \mathrm{S}$) and the orchestrator does not hide it ($\langle \alpha, \overline{\alpha} \rangle \in \mathrm{A}$); symmetrically for the service.

With these notions we can now define the coinductive characterization of weak subcontract relation, in a similar manner as for the strong variant.

Definition 3.7 (coinductive weak subcontract)**.** We say that $\mathscr{W}_k$ is a *coinductive weak k-subcontract relation* if $(\mathbb{B}, \sigma, \tau) \in \mathscr{W}_k$ implies that $\mathbb{B}$ is a k-buffer and there exists a set of orchestration actions A such that $\mathbb{B} \vdash_k \mathrm{A}$ and

(1) $\tau \Downarrow \mathrm{S}$ implies either ($\sigma \Downarrow \mathrm{R}$ and $\mathrm{R} \subseteq \mathrm{A} \circ \mathrm{S}$ for some R) or $(\emptyset \bullet \mathrm{A}) \cap \overline{\mathrm{S}} \neq \emptyset$, and
(2) $\tau \stackrel{\varphi'}{\Longrightarrow}$ and $\langle \varphi, \overline{\varphi}' \rangle \in \mathrm{A}$ implies $\sigma \stackrel{\varphi}{\Longrightarrow}$ and $(\mathbb{B}\langle \varphi, \overline{\varphi}' \rangle, \sigma(\varphi), \tau(\varphi')) \in \mathscr{W}_k$.

We write $\sigma \preceq^{\mathsf{c}} \tau$ if there exists $\mathscr{W}_k$ such that $(\tilde{\emptyset}, \sigma, \tau) \in \mathscr{W}_k$.

Condition (1) requires that either τ can be made more deterministic than σ by means of the orchestrator (the ready set $\mathrm{A} \circ \mathrm{S}$ of the orchestrated service has a corresponding one of σ that offers fewer actions), or that τ can be satisfied by the orchestrator without any help from the client ($(\emptyset \bullet \mathrm{A}) \cap \overline{\mathrm{S}} \neq \emptyset$ implies that $\langle \varepsilon, \overline{\alpha} \rangle \in \mathrm{A}$ and $\alpha \in \mathrm{S}$ for some α). Condition (2) poses the usual requirement that the continuations must be in the subcontract relation.

The two definitions of weak subcontract are equivalent:

Theorem 3.8. $\preceq = \preceq^{\mathsf{c}}$.

Proof. ($\preceq_k \subseteq \preceq_k^{\mathsf{c}}$) Let $f : \sigma \preceq_k \tau$ and assume, without loss of generality, that f is relevant for $\sigma \preceq_k \tau$. It is sufficient to prove that

$$\mathscr{W}_k \stackrel{\text{def}}{=} \{(\tilde{\emptyset}\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle, \sigma(\varphi_1 \cdots \varphi_n), \tau(\varphi'_1 \cdots \varphi'_n)) \mid f \xmapsto{\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle}\}$$

is a coinductive k-subcontract relation. Let $(\mathbb{B}, \sigma', \tau') \in \mathscr{W}_k$. Then there exist $\varphi_1, \ldots, \varphi_n$ and $\varphi'_1, \ldots, \varphi'_n$ and f' such that $f \xmapsto{\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle} f'$, $\mathbb{B} = \tilde{\emptyset}\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle$, $\sigma' = \sigma(\varphi_1 \cdots \varphi_n)$, and $\tau' = \tau(\varphi'_1 \cdots \varphi'_n)$. Since f is a k-orchestrator we have that $\mathbb{B}$ is a k-buffer. Let $\mathrm{A} \stackrel{\text{def}}{=} \mathtt{init}(f')$. As regards condition (1) of Definition 3.7, let $\mathrm{R}_1, \ldots, \mathrm{R}_m$ be the ready sets of σ'. Assume

by contradiction that there exists S such that $\tau' \Downarrow \mathrm{S}$ and $\mathrm{R}_i \not\subseteq \mathrm{A} \circ \mathrm{S}$ for every $1 \leq i \leq m$ and $(\emptyset \bullet \mathrm{A}) \cap \overline{\mathrm{S}} = \emptyset$. Then there exists $\alpha_i \in \mathrm{R}_i \setminus \mathrm{A} \circ \mathrm{S}$ for every $1 \leq i \leq m$. Let $\rho \stackrel{\text{def}}{=} \sum_{1 \leq i \leq m} \overline{\alpha}_i.\mathsf{e}$. We have $\rho \dashv \sigma'$ but $f' : \rho \not\dashv \tau'$, which is absurd. As regards condition (2) of Definition 3.7, assume $\tau' \stackrel{\varphi'}{\Longrightarrow}$ and $\langle \varphi, \overline{\varphi}' \rangle \in \mathrm{A}$. Since f is relevant we have $\sigma' \stackrel{\varphi}{\Longrightarrow}$. We conclude $(\mathbb{B}\langle \varphi, \overline{\varphi}' \rangle, \sigma'(\varphi), \tau'(\varphi')) \in \mathscr{W}_k$ by definition of $\mathscr{W}_k$.

($\preceq_k^{\mathsf{c}} \subseteq \preceq_k$) Let $\mathscr{W}_k$ be a coinductive weak k-subcontract relation such that $(\tilde{\emptyset}, \sigma, \tau) \in \mathscr{W}_k$ and assume $\rho \dashv \sigma$. Let $\mathrm{A}(\mathbb{B}, \sigma', \tau')$ stand for the set A of orchestration actions satisfying conditions (1) and (2) of Definition 3.7 whenever $(\mathbb{B}, \sigma', \tau') \in \mathscr{W}_k$. Let

$$f(\mathbb{B}, \sigma', \tau') \stackrel{\text{def}}{=} \bigvee_{\langle \varphi, \varphi' \rangle \in \mathrm{A}(\mathbb{B}, \sigma', \tau')} \langle \varphi, \varphi' \rangle.f(\mathbb{B}\langle \varphi, \overline{\varphi}' \rangle, \sigma'(\varphi), \tau'(\varphi'))$$

and let $f \stackrel{\text{def}}{=} f(\tilde{\emptyset}, \sigma, \tau)$. Observe that f is well defined by regularity of σ and τ and that it is a k-orchestrator. We prove $f : \rho \dashv\!\vdash \tau$. Consider a derivation $\rho \|_f \tau \Longrightarrow \rho' \|_{f'} \tau' \not\longrightarrow$. By "unzipping" this derivation we obtain that there exist $\varphi_1, \ldots, \varphi_n$ and $\varphi'_1, \ldots, \varphi'_n$ such that $\rho \stackrel{\overline{\varphi}_1 \cdots \overline{\varphi}_n}{\Longrightarrow} \rho' \not\longrightarrow$ and $f \stackrel{\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle}{\longmapsto\!\!\!\!\!\!\longrightarrow} f'$ and $\tau \stackrel{\varphi'_1 \cdots \varphi'_n}{\Longrightarrow} \tau' \not\longrightarrow$. By condition (2) of Definition 3.7 and by induction on n we derive that $\sigma \stackrel{\varphi_1 \cdots \varphi_n}{\Longrightarrow}$ and $(\tilde{\emptyset}\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle, \sigma(\varphi_1 \cdots \varphi_n), \tau(\varphi'_1 \cdots \varphi'_n)) \in \mathscr{W}_k$. Observe that $\tau(\varphi'_1 \cdots \varphi'_n) \Downarrow \mathtt{init}(\tau')$. By condition (1) of Definition 3.7 we have that either there exists R such that $\sigma(\varphi_1 \cdots \varphi_n) \Downarrow \mathrm{R}$ and $\mathrm{R} \subseteq \mathtt{init}(f') \circ \mathtt{init}(\tau')$ or $(\emptyset \bullet \mathtt{init}(f')) \cap \overline{\mathtt{init}(\tau')} \neq \emptyset$. However from $\rho' \|_{f'} \tau' \not\longrightarrow$ we derive $(\emptyset \bullet \mathtt{init}(f')) \cap \overline{\mathtt{init}(\tau')} = \emptyset$, hence there exists σ' such that $\sigma \stackrel{\varphi_1 \cdots \varphi_n}{\Longrightarrow} \sigma'$ and $\mathtt{init}(\sigma') \subseteq \mathtt{init}(f') \circ \mathtt{init}(\tau')$. By "zipping" the derivations starting from ρ and σ we obtain $\rho \parallel \sigma \Longrightarrow \rho' \parallel \sigma'$. Furthermore $\rho' \parallel \sigma' \not\longrightarrow$ because $\mathtt{init}(\sigma') \subseteq \mathtt{init}(f') \circ \mathtt{init}(\tau')$. From $\rho \dashv \sigma$ we conclude $\rho' \stackrel{\mathsf{e}}{\longrightarrow}$. □

3.4 Orchestrators as morphisms

When $f : \sigma \preceq \tau$ *every* client that is strongly compliant with σ is also weakly compliant with τ by means of the orchestrator f. In a sense, it is as if the orchestrator f transforms the service with contract τ into a service with contract σ. The function determined by an orchestrator can be effectively computed as by the following definition.

Definition 3.9 (orchestrator application)**.** The *application* of the orchestrator

f to the (service) contract σ, notation $f(\sigma)$, is defined as

$$f(\sigma) \stackrel{\text{def}}{=} \bigoplus_{\sigma \Downarrow \mathrm{R}} \begin{cases} \sum_{f \stackrel{\langle \alpha, \varepsilon \rangle}{\longmapsto} f'} \alpha.f'(\sigma) + \sum_{f \stackrel{\langle \alpha, \overline{\alpha} \rangle}{\longmapsto} f', \alpha \in \mathrm{R}} \alpha.f'(\sigma(\alpha)) & \\ \qquad\qquad \text{if } (\emptyset \bullet \mathtt{init}(f)) \cap \overline{\mathrm{R}} = \emptyset & \\ \Big(\Big(\sum_{f \stackrel{\langle \alpha, \varepsilon \rangle}{\longmapsto} f'} \alpha.f'(\sigma) + \sum_{f \stackrel{\langle \alpha, \overline{\alpha} \rangle}{\longmapsto} f', \alpha \in \mathrm{R}} \alpha.f'(\sigma(\alpha)) \Big) \oplus \mathtt{0} \Big) & \\ \quad + \bigoplus_{f \stackrel{\langle \varepsilon, \overline{\alpha} \rangle}{\longmapsto} f', \alpha \in \mathrm{R}} f'(\sigma(\alpha)) \qquad \text{otherwise} & \end{cases}$$

The equation reminds of the *expansion law* for the parallel operator in full CCS [26], but describing the interaction of the orchestrator and the service. The first line defines the behavior of the orchestrated service when no synchronization between orchestrator and service occurs ($(\emptyset \bullet \mathtt{init}(f)) \cap \overline{\mathrm{R}} = \emptyset$): all the asynchronous orchestration actions are available, in addition to all the synchronous orchestration actions that are enabled by the service contract when in state R. In the second line there is at least one asynchronous orchestration action that can synchronize with the service in state R ($(\emptyset \bullet \mathtt{init}(f)) \cap \overline{\mathrm{R}} \neq \emptyset$). In this case the client perceives an appropriate combination of actions among those that are available before and after the synchronization occurs. The internal choice with the null summand indicates that actions available before the synchronization are not guaranteed (if the synchronization does actually occur), whereas all the actions after the synchronization are (the client can just wait for the orchestrator and the service to reach a stable state). As an example consider $f \stackrel{\text{def}}{=} \langle a, \varepsilon \rangle.\langle c, \varepsilon \rangle.(\langle \varepsilon, \overline{a} \rangle.\langle \overline{b}, b \rangle \vee \langle \varepsilon, \overline{c} \rangle.\langle \overline{d}, d \rangle)$. Then:

- $f(a.\overline{b}) = a.c.\overline{b}$;
- $f(a.\overline{b} + c.\overline{d}) = a.c.(\overline{b} \oplus \overline{d})$;
- $f(a.\overline{b} \oplus c.\overline{d}) = a.c.(\mathtt{0} \oplus \overline{b} \oplus \overline{d})$.

In general we have $\langle \alpha, \overline{\alpha} \rangle.f(\alpha.\sigma) = \alpha.f(\sigma)$ and $\langle \alpha, \varepsilon \rangle.f(\sigma) = \alpha.f(\sigma)$ and $\langle \varepsilon, \overline{\alpha} \rangle.f(\alpha.\sigma) = f(\sigma)$.

Well-foundedness of Definition 3.9 is a direct consequence of the regularity of f and σ. Indeed, one should think of $f(\sigma)$ as a label associated with the contract on the r.h.s. of the definition. Then, we see that $f(\sigma)$ depends on a finite number of $f'(\sigma(\alpha))$ where $f \stackrel{\mu}{\longmapsto} f'$ and each of these, in turn, depends on a finite number of $f''(\sigma(\alpha)(\beta))$ where $f' \stackrel{\mu'}{\longmapsto} f''$. Regularity of f and σ and Proposition 2.4 assure us that we only need a finite number of labels $f(\sigma)$ that one can then fold using recursive terms in the usual way.

The next result proves that $f(\sigma)$ is indeed the contract of the orchestrated service, namely it satisfies the same clients that are weakly compliant with σ by means of f:

Theorem 3.10. *$f : \rho \dashv\!\mid \sigma$ if and only if $\rho \dashv f(\sigma)$.*

Proof. ($\Rightarrow$) Assume $f : \rho \dashv\!\shortmid \sigma$ and consider a derivation $\rho \parallel f(\sigma) \Longrightarrow \rho' \parallel \tau \not\longrightarrow$. Then there exist φ, φ', and f' such that $\rho \stackrel{\overline{\varphi}}{\Longrightarrow} \rho' \not\longrightarrow$ and $f(\sigma) \stackrel{\varphi}{\Longrightarrow} f'(\sigma(\varphi')) \Longrightarrow \tau \not\longrightarrow$. By definition of $f(\sigma)$ there exist $\varphi_1, \ldots, \varphi_n$ and $\varphi'_1, \ldots, \varphi'_n$ such that $\varphi = \varphi_1 \cdots \varphi_n$ and $\varphi' = \varphi'_1 \cdots \varphi'_n$ and $f \stackrel{\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle}{\longmapsto} f'$. From $\rho' \parallel \tau \not\longrightarrow$ we deduce $\overline{\mathtt{init}(\rho')} \cap \mathtt{init}(\tau) = \emptyset$. From $f'(\sigma(\varphi')) \Longrightarrow \tau \not\longrightarrow$ and by definition of $f'(\sigma(\varphi'))$ we deduce that there exist $\alpha_1, \ldots, \alpha_m$ and σ' and f'' such that $\sigma \stackrel{\varphi' \alpha_1 \cdots \alpha_m}{\Longrightarrow} \sigma' \not\longrightarrow$ and $f' \stackrel{\langle \varepsilon, \overline{\alpha}_1 \rangle \cdots \langle \varepsilon, \overline{\alpha}_m \rangle}{\longmapsto} f''$ and $(\emptyset \bullet \mathtt{init}(f'')) \cap \overline{\mathtt{init}(\sigma')} = \emptyset$ and $\mathtt{init}(f'') \bullet \mathtt{init}(\sigma') \subseteq \mathtt{init}(\tau)$. By zipping the derivations starting from ρ, f, and σ we obtain $\rho \|_f \sigma \Longrightarrow \rho' \|_{f''} \sigma'$. Furthermore $\rho' \|_{f''} \sigma' \not\longrightarrow$ because $\rho' \not\longrightarrow$ and $\sigma' \not\longrightarrow$ and $\overline{\mathtt{init}(\rho')} \cap (\mathtt{init}(f'') \bullet \mathtt{init}(\sigma')) \subseteq \overline{\mathtt{init}(\rho')} \cap \mathtt{init}(\tau) = \emptyset$. From $f : \rho \dashv\!\shortmid \sigma$ we conclude $\rho' \stackrel{\mathsf{e}}{\longrightarrow}$.

($\Leftarrow$) Assume $\rho \dashv f(\sigma)$ and consider a derivation $\rho \|_f \sigma \Longrightarrow \rho' \|_{f'} \sigma' \not\longrightarrow$. By "unzipping" this derivation we have that there exist $\varphi_1, \ldots, \varphi_n$ and $\varphi'_1, \ldots, \varphi'_n$ such that $\rho \stackrel{\overline{\varphi}_1 \cdots \overline{\varphi}_n}{\Longrightarrow} \rho' \not\longrightarrow$ and $f \stackrel{\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle}{\longmapsto} f'$ and $\sigma \stackrel{\varphi'_1 \cdots \varphi'_n}{\Longrightarrow} \sigma' \not\longrightarrow$. Furthermore from $\rho' \|_{f'} \sigma' \not\longrightarrow$ we derive $\overline{\mathtt{init}(\rho')} \cap (\mathtt{init}(f') \bullet \mathtt{init}(\sigma')) = \emptyset$. By definition of $f(\sigma)$ there exists τ such that $f(\sigma) \stackrel{\varphi_1 \cdots \varphi_n}{\Longrightarrow} \tau \not\longrightarrow$ and $\mathtt{init}(\tau) = \mathtt{init}(f') \bullet \mathtt{init}(\sigma')$. By zipping the derivations starting from ρ and $f(\sigma)$ we obtain $\rho \parallel f(\sigma) \Longrightarrow \rho' \parallel \tau$. Furthermore $\rho' \parallel \tau \not\longrightarrow$ because $\rho' \not\longrightarrow$, $\tau \not\longrightarrow$, and $\overline{\mathtt{init}(\rho')} \cap \mathtt{init}(\tau) = \overline{\mathtt{init}(\rho')} \cap (\mathtt{init}(f') \bullet \mathtt{init}(\sigma')) = \emptyset$. From $\rho \dashv f(\sigma)$ we conclude $\rho' \stackrel{\mathsf{e}}{\longrightarrow}$. □

We are now able to connect the strong and weak subcontract relations.

Corollary 3.11. *$f : \sigma \preceq \tau$ if and only if $\sigma \sqsubseteq f(\tau)$.*

Proof. By Theorem 3.10 $f : \sigma \preceq \tau$ if and only if $\rho \dashv \sigma$ implies $f : \rho \dashv\!\shortmid \tau$ if and only if $\rho \dashv \sigma$ implies $\rho \dashv f(\sigma)$ if and only if $\sigma \sqsubseteq f(\tau)$. □

Corollary 3.11 also provides us with a handy tool for studying the properties of $\preceq$ since we can reduce the weak subcontract relation $\preceq$ to the more familiar strong subcontract relation $\sqsubseteq$. For example, we can reduce checking $f : \sigma \preceq \tau$ to checking $\sigma \sqsubseteq f(\tau)$ by computing $f(\tau)$. The next few examples show that $\preceq$ includes $\sqsubseteq$, that $\preceq$ permits width and depth extensions and, to some extent, permutation of actions:

- $a \oplus b \preceq a$ since $a \oplus b \sqsubseteq a = \langle a, \overline{a} \rangle(a)$;
- $a \preceq a + b$ since $a = \langle a, \overline{a} \rangle(a + b)$;
- $a \preceq a.b$ since $a = \langle a, \overline{a} \rangle(a.b)$;
- $\overline{a}.\overline{c}.b \preceq \overline{c}.\overline{a}.b$ since $\overline{a}.\overline{c}.b = \langle \varepsilon, c \rangle.\langle \overline{a}, a \rangle.\langle \overline{c}, \varepsilon \rangle.\langle \overline{b}, b \rangle(\overline{c}.\overline{a}.b)$;
- $a.c.\overline{b} \preceq c.a.\overline{b}$ since $a.c.\overline{b} = \langle a, \varepsilon \rangle.\langle c, \overline{c} \rangle.\langle \varepsilon, \overline{a} \rangle.\langle b, \overline{b} \rangle(c.a.\overline{b})$.

As regards permutations, it is possible in general to postpone input actions.

For instance we have $a.\beta.\sigma \preceq \beta.a.\sigma$ where the service on the r.h.s. of $\preceq$ is able to perform the β action without having performed a first. On the other hand, we have $\overline{a}.b.\sigma \not\preceq b.\overline{a}.\sigma$ because no valid orchestrator is capable of sending an $\overline{a}$ message to the client, without having received it in advance from the service. The fact that this relation does not hold is reasonable since a client of the service on the l.h.s. may need the information contained in the $\overline{a}$ message before sending the b message back to the service.

The morphism induced by an orchestrator f is monotone with respect to the strong subcontract relation and is well behaved with respect to the choice operators.

Proposition 3.12. *The following properties hold:*

(1) $\sigma \sqsubseteq \tau$ *implies* $f(\sigma) \sqsubseteq f(\tau)$*;*
(2) $f(\sigma) + f(\tau) \sqsubseteq f(\sigma + \tau)$*;*
(3) $f(\sigma) \oplus f(\tau) \simeq f(\sigma \oplus \tau)$*.*

Proof. We prove item (1); items (2) and (3) are similar. By Theorem 3.10 it is sufficient to prove that $f : \rho \dashv\!\shortmid \sigma$ implies $f : \rho \dashv\!\shortmid \tau$ for every ρ, under the hypothesis $\sigma \sqsubseteq \tau$. Consider a derivation $\rho \|_f \tau \Longrightarrow \rho' \|_{f'} \tau' \not\longrightarrow$. Then there exist $\varphi_1, \ldots, \varphi_n$ and $\varphi'_1, \ldots, \varphi'_n$ such that $\rho \xRightarrow{\overline{\varphi}_1 \cdots \overline{\varphi}_n} \rho' \not\longrightarrow$ and $f \xmapsto{\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle} f'$ and $\tau \xRightarrow{\varphi'_1 \cdots \varphi'_n} \tau' \not\longrightarrow$. Furthermore, from $\rho' \|_{f'} \tau' \not\longrightarrow$ we deduce $\overline{\mathtt{init}(\rho')} \cap (\mathtt{init}(f') \bullet \mathtt{init}(\tau')) = \emptyset$. From $\sigma \sqsubseteq \tau$ we derive that there exists σ' such that $\sigma \xRightarrow{\varphi'_1 \cdots \varphi'_n} \sigma' \not\longrightarrow$ and $\mathtt{init}(\sigma') \subseteq \mathtt{init}(\tau')$. By zipping the derivations starting from ρ, f, and σ we obtain $\rho \|_f \sigma \Longrightarrow \rho' \|_{f'} \sigma'$. Furthermore, $\overline{\mathtt{init}(\rho')} \cap (\mathtt{init}(f') \bullet \mathtt{init}(\sigma')) \subseteq \overline{\mathtt{init}(\rho')} \cap (\mathtt{init}(f') \bullet \mathtt{init}(\tau')) = \emptyset$, hence $\rho' \|_{f'} \sigma' \not\longrightarrow$. From $f : \rho \dashv\!\shortmid \sigma$ we conclude $\rho' \xrightarrow{\mathsf{e}}$. □

Observe that $f(\sigma) + f(\tau) \simeq f(\sigma + \tau)$ does not hold in general, because of the asynchronous actions that f may offer to the client side. Consider for example $f \stackrel{\text{def}}{=} \langle a, \varepsilon \rangle.(\langle \overline{b}, b \rangle + \langle \overline{d}, d \rangle)$. Then $f(\overline{b}) + f(\overline{d}) = a.\overline{b} + a.\overline{d} \simeq a.(\overline{b} \oplus \overline{d}) \sqsubseteq a.(\overline{b} + \overline{d}) = f(\overline{b} + \overline{d})$ but $f(\overline{b} + \overline{d}) \not\sqsubseteq f(\overline{b}) + f(\overline{d})$. Nonetheless Proposition 3.12, in conjunction with Proposition 2.7, allows us to prove an interesting property of $\preceq$: if $\sigma \sqsubseteq f(\sigma')$ and $\tau \sqsubseteq f(\tau')$, then $\sigma + \tau \sqsubseteq f(\sigma' + \tau')$ and $\sigma \oplus \tau \sqsubseteq f(\sigma' \oplus \tau')$. This means that if $\sigma \preceq \sigma'$ and $\tau \preceq \tau'$ and the two relations are witnessed by the *same* orchestrator, then $\sigma + \tau \preceq \sigma' + \tau'$ and $\sigma \oplus \tau \preceq \sigma' \oplus \tau'$. In other words, a sufficient condition for being able to orchestrate $\sigma' + \tau'$ is that the orchestrator must be independent of the branch (either σ' or τ') taken by the service, which is in fact the minimum requirement we could expect. In general however $\preceq$ is not a precongruence: $a \preceq a + b.c$ but $a + b.d \not\preceq a + b.c + b.d \simeq a + b.(c \oplus d)$.

Transitivity of the weak subcontract relation is not granted by the definition of $\preceq$, because $\sigma \preceq \tau$ means that every client that is *strongly* compliant with σ is also *weakly* compliant with τ. So it is not clear whether $\sigma \preceq \tau$ and $\tau \preceq \sigma'$ implies $\sigma \preceq \sigma'$. Observe that transitivity of $\preceq$ is necessary in order to enhance Web service discovery as described in §1.

Let us start from the hypotheses $f : \sigma \preceq \tau$ and $g : \tau \preceq \sigma'$. By Corollary 3.11 we know that $\sigma \sqsubseteq f(\tau)$ and $\tau \sqsubseteq g(\sigma')$. Furthermore, by Proposition 3.12(1) and transitivity of $\sqsubseteq$ we deduce that $\sigma \sqsubseteq f(\tau) \sqsubseteq f(g(\sigma'))$. Thus we can conclude $\sigma \preceq \sigma'$ provided that for any two orchestrators f and g their functional composition $f \circ g$ is still an orchestrator. This is not the case in general. To see why, consider for example

$$f \stackrel{\text{def}}{=} \langle a, \varepsilon\rangle.\langle c, \varepsilon\rangle.(\langle \varepsilon, \overline{a}\rangle.\langle \overline{b}, b\rangle \vee \langle \varepsilon, \overline{c}\rangle.\langle \overline{d}, d\rangle) \quad \text{and} \quad g \stackrel{\text{def}}{=} \langle a, \varepsilon\rangle.\langle \overline{b}, b\rangle \vee \langle c, \varepsilon\rangle.\langle \overline{d}, d\rangle$$

and apply them to the contract $\sigma \stackrel{\text{def}}{=} \overline{b} + \overline{d}$. We obtain

$$f(g(\sigma)) \simeq f(a.\overline{b} + c.\overline{d}) \simeq a.c.(\overline{b} \oplus \overline{d})$$

The subsequent applications of g first and then f introduce some nondeterminism due to the uncertainty as to which synchronization (on a or on c) will occur. This uncertainty yields the internal choice $\overline{b} \oplus \overline{d}$ in the resulting contract. No single orchestrator can turn $\overline{b} + \overline{d}$ into $a.c.(\overline{b} \oplus \overline{d})$ for orchestrators do not manifest internal nondeterminism. The problem could be addressed by adding internal nondeterminism to the orchestration language, but this seems quite artificial and, as a matter of facts, is unnecessary. If we are able to find an orchestrator $f \cdot g$ such that $(f \circ g)(\sigma') \sqsubseteq (f \cdot g)(\sigma')$, then $\sigma \sqsubseteq (f \cdot g)(\sigma')$ follows by transitivity of $\sqsubseteq$.

Definition 3.13 (orchestrator composition)**.** The *composition* of two orchestrators f and g, notation $f \cdot g$, is defined as:

$$\begin{aligned} f \cdot g \stackrel{\text{def}}{=} & \bigvee_{f \stackrel{\langle \alpha, \varepsilon\rangle}{\longmapsto} f'} \langle \alpha, \varepsilon\rangle.(f' \cdot g) \vee \bigvee_{g \stackrel{\langle \varepsilon, \overline{\alpha}\rangle}{\longmapsto} g'} \langle \varepsilon, \overline{\alpha}\rangle.(f \cdot g') \\ & \vee \bigvee_{f \stackrel{\langle \varphi, \overline{\alpha}\rangle}{\longmapsto} f', g \stackrel{\langle \alpha, \varphi'\rangle}{\longmapsto} g', \varphi\varphi' \neq \varepsilon} \langle \varphi, \varphi'\rangle.(f' \cdot g') \vee \bigvee_{f \stackrel{\langle \varepsilon, \overline{\alpha}\rangle}{\longmapsto} f', g \stackrel{\langle \alpha, \varepsilon\rangle}{\longmapsto} g'} (f' \cdot g') \end{aligned}$$

The first two subterms in the definition of $f \cdot g$ indicate that all the asynchronous actions offered by f (respectively, g) to the client (respectively, service) are available. The third subterm turns synchronous actions into asynchronous ones: for example, $\langle \alpha, \overline{\alpha}\rangle \cdot \langle \alpha, \varepsilon\rangle = \langle \alpha, \varepsilon\rangle$ and $\langle \varepsilon, \overline{\alpha}\rangle \cdot \langle \alpha, \overline{\alpha}\rangle = \langle \varepsilon, \alpha\rangle$.

The last subterm accounts for the "synchronizations" occurring within the orchestrator, when f and g exchange a message and the two actions annihilate each other. If we consider the orchestrators f and g defined above, we obtain $f \cdot g = \langle a, \varepsilon\rangle.\langle c, \varepsilon\rangle.(\langle \overline{b}, b\rangle \vee \langle \overline{d}, d\rangle)$ and we observe $(f \cdot g)(\overline{b} + \overline{d}) = a.c.(\overline{b} + \overline{d})$.

Well-foundedness of $f \cdot g$ can be determined in a similar way as has been done for $f(\sigma)$. The proof that $f \cdot g$ is the orchestrator we are looking for needs the following technical result, which tells us about the "unzipping" of compound orchestrators.

Lemma 3.14. *$f \cdot g \xmapsto{\langle \psi_1, \overline{\psi}'_1\rangle \cdots \langle \psi_m, \overline{\psi}'_m\rangle} h$ implies that there exist $\varphi_1, \ldots, \varphi_n$ and $\varphi'_1, \ldots, \varphi'_n$ and $\varphi''_1, \ldots, \varphi''_n$ such that $f \xmapsto{\langle \varphi_1, \overline{\varphi}'_1\rangle \cdots \langle \varphi_n, \overline{\varphi}'_n\rangle} f'$ and $g \xmapsto{\langle \varphi'_1, \overline{\varphi}''_1\rangle \cdots \langle \varphi'_n, \overline{\varphi}''_n\rangle} g'$ and $\psi_1 \cdots \psi_m = \varphi_1 \cdots \varphi_n$ and $\psi'_1 \cdots \psi'_m = \varphi''_1 \cdots \varphi''_n$ and $\mathtt{init}(f' \cdot g') \subseteq \mathtt{init}(h)$.*

Proof. In this proof we adopt the following notation: we write $f \xhookrightarrow{\langle \alpha_1 \cdots \alpha_n, \varepsilon\rangle} f'$ if $f \xmapsto{\langle \alpha_1, \varepsilon\rangle \cdots \langle \alpha_n, \varepsilon\rangle} f'$ and $f \xhookrightarrow{\langle \varepsilon, \alpha_1 \cdots \alpha_n\rangle} f'$ if $f \xmapsto{\langle \varepsilon, \alpha_1\rangle \cdots \langle \varepsilon, \alpha_n\rangle} f'$. We admit $n = 0$, in which case we have $f \xhookrightarrow{\langle \varepsilon, \varepsilon\rangle} f$. We prove the result for $m = 1$. The general statement follows by a simple induction on m. Assume $f \cdot g \xmapsto{\langle \psi, \overline{\psi}'\rangle} h$. Then

$$h = \bigvee_{\substack{f \xhookrightarrow{\langle \varepsilon, \overline{\varphi}\rangle} f' \\ g \xhookrightarrow{\langle \varphi, \varepsilon\rangle} g'}} \left(\bigvee_{\substack{f' \xmapsto{\langle \psi, \varepsilon\rangle} f'' \\ \psi' = \varepsilon}} f'' \cdot g' \vee \bigvee_{\substack{g' \xmapsto{\langle \varepsilon, \overline{\psi}'\rangle} g'' \\ \psi = \varepsilon}} f' \cdot g'' \vee \bigvee_{\substack{f' \xmapsto{\langle \psi, \overline{\alpha}\rangle} f'' \\ g' \xmapsto{\langle \alpha, \overline{\psi}'\rangle} g''}} f'' \cdot g'' \right)$$

namely h accounts for all the possible continuations of the action $\langle \psi, \overline{\psi}'\rangle$ considering all the possible "synchronizations" occurring within $f \cdot g$. All these synchronizations are captured by iterating over all φ such that $f \xhookrightarrow{\langle \varepsilon, \overline{\varphi}\rangle} f'$ and $g \xhookrightarrow{\langle \varphi, \varepsilon\rangle} g'$. There is a finite number of them because f and g are valid orchestrators of finite rank. We deduce that there exist $\varphi'_1, \ldots, \varphi'_n$ such that

$$f \xmapsto{\langle \varepsilon, \overline{\varphi}'_1\rangle \cdots \langle \varepsilon, \overline{\varphi}'_{n-1}\rangle \langle \psi, \overline{\varphi}'_n\rangle} f' \qquad \text{and} \qquad g \xmapsto{\langle \varphi'_1, \varepsilon\rangle \cdots \langle \varphi'_{n-1}, \varepsilon\rangle \langle \varphi'_n, \psi'\rangle} g'$$

and we conclude by taking $\varphi_1 = \cdots = \varphi_{n-1} = \varphi''_1 = \cdots = \varphi''_{n-1} = \varepsilon$ and $\varphi_n = \psi$ and $\varphi''_n = \psi'$. The fact that $\mathtt{init}(f' \cdot g') \subseteq \mathtt{init}(h)$ is an immediate consequence of the fact that $f' \cdot g'$ is a summand occurring in h. □

The next result proves that $f \cdot g$ is correct and, as a corollary, that $\preceq$ is a preorder:

Theorem 3.15. *$f(g(\sigma)) \sqsubseteq (f \cdot g)(\sigma)$.*

Proof. Let $\rho \dashv f(g(\sigma))$. By Theorem 3.10 it is sufficient to show that $f \cdot g : \rho \dashv\vdash \sigma$, so consider a derivation $\rho \parallel_{f \cdot g} \sigma \Longrightarrow \rho' \parallel_h \sigma' \not\rightarrow$. By unzipping this derivation we deduce that there exist $\psi_1, \ldots, \psi_m$ and $\psi'_1, \ldots, \psi'_m$ such that $\rho \xRightarrow{\overline{\psi}_1 \cdots \overline{\psi}_m} \rho' \not\rightarrow$ and $f \cdot g \xmapsto{\langle \psi_1, \overline{\psi}'_1 \rangle \cdots \langle \psi_m, \overline{\psi}'_m \rangle} h$ and $\sigma \xRightarrow{\psi'_1 \cdots \psi'_m} \sigma' \not\rightarrow$. From $\rho' \parallel_h \sigma' \not\rightarrow$ we deduce $\overline{\mathtt{init}(\rho')} \cap (\mathtt{init}(h) \bullet \mathtt{init}(\sigma')) = \emptyset$. By Lemma 3.14 we derive that there exist $\varphi_1, \ldots, \varphi_n$, $\varphi'_1, \ldots, \varphi'_n$, and $\varphi''_1, \ldots, \varphi''_n$ such that $f \xmapsto{\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle} f'$ and $g \xmapsto{\langle \varphi'_1, \overline{\varphi}''_1 \rangle \cdots \langle \varphi'_n, \overline{\varphi}''_n \rangle} g'$ and $\psi_1 \cdots \psi_m = \varphi_1 \cdots \varphi_n$ and $\psi'_1 \cdots \psi'_m = \varphi''_1 \cdots \varphi''_n$ and $\mathtt{init}(f' \cdot g') \subseteq \mathtt{init}(h)$. Since f and g are valid orchestrators of finite rank, there exist f'', g'', and φ such that $f' \xhookrightarrow{\langle \varepsilon, \overline{\varphi} \rangle} f''$ and $g' \xhookrightarrow{\langle \varphi, \varepsilon \rangle} g''$ and $f'' \xmapsto{\langle \varepsilon, \overline{\alpha} \rangle}$ implies $g'' \xmapsto{\langle \alpha, \varepsilon \rangle}$. Namely f'' and g'' are two residual orchestrators that do not "synchronize" with each other. By definition of orchestrator composition, observe that $\mathtt{init}(f'' \cdot g'') \subseteq \mathtt{init}(f' \cdot g')$ because $f'' \cdot g''$ is a summand within $f' \cdot g'$. By definition of orchestrator application we have $g(\sigma) \xRightarrow{\varphi'_1 \cdots \varphi'_n} g'(\sigma(\varphi''_1 \cdots \varphi''_n)) \Longrightarrow g'(\sigma') \xRightarrow{\varphi} g''(\sigma')$. Furthermore $\emptyset \circ \mathtt{init}(g'') \subseteq \emptyset \circ \mathtt{init}(f'' \cdot g'') \subseteq \emptyset \circ \mathtt{init}(f' \cdot g') \subseteq \emptyset \circ \mathtt{init}(h)$ hence $(\emptyset \circ \mathtt{init}(g'')) \cap \overline{\mathtt{init}(\sigma')} = \emptyset$ and $g''(\sigma') \not\rightarrow$ and $\mathtt{init}(g''(\sigma)) = \mathtt{init}(g'') \bullet \mathtt{init}(\sigma')$. By definition of orchestrator application we have $f(g(\sigma)) \xRightarrow{\varphi_1 \cdots \varphi_n} f'(g(\sigma)(\varphi'_1 \cdots \varphi'_n)) = f'(g'(\sigma(\varphi''_1 \cdots \varphi''_n))) \Longrightarrow f'(g'(\sigma')) \Longrightarrow f''(g''(\sigma'))$. Now we want to show that $(\emptyset \circ \mathtt{init}(f'')) \cap \overline{\mathtt{init}(g''(\sigma'))} = \emptyset$. From $\mathtt{init}(g''(\sigma')) = \mathtt{init}(g'') \bullet \mathtt{init}(\sigma')$ we derive that $(\emptyset \circ \mathtt{init}(f'')) \cap \overline{\mathtt{init}(g''(\sigma'))} \neq \emptyset$ if and only if there exists α such that $f'' \xmapsto{\langle \varepsilon, \overline{\alpha} \rangle}$ and either $g'' \xmapsto{\langle \alpha, \varepsilon \rangle}$ or ($g'' \xmapsto{\langle \alpha, \overline{\alpha} \rangle}$ and $\sigma' \xrightarrow{\alpha}$). However, by the way f'' and g'' have been chosen we have that $f'' \xmapsto{\langle \varepsilon, \overline{\alpha} \rangle}$ implies $g'' \xmapsto{\langle \alpha, \varepsilon \rangle}$. Furthermore, if $f'' \xmapsto{\langle \varepsilon, \overline{\alpha} \rangle}$ and $g'' \xmapsto{\langle \alpha, \overline{\alpha} \rangle}$, then $\langle \varepsilon, \overline{\alpha} \rangle \in \mathtt{init}(f'' \cdot g'') \subseteq \mathtt{init}(h)$. Then $\sigma' \xrightarrow{\alpha}$ because $\rho' \parallel_h \sigma' \not\rightarrow$. Hence $(\emptyset \circ \mathtt{init}(f'')) \cap \overline{\mathtt{init}(g''(\sigma'))} = \emptyset$, so $f''(g''(\sigma')) \not\rightarrow$ and $\mathtt{init}(f''(g''(\sigma'))) = \mathtt{init}(f'') \bullet \mathtt{init}(g''(\sigma')) = \mathtt{init}(f'') \bullet \mathtt{init}(g'') \bullet \mathtt{init}(\sigma') = \mathtt{init}(f'' \cdot g'') \bullet \mathtt{init}(\sigma') \subseteq \mathtt{init}(h) \bullet \mathtt{init}(\sigma')$. By zipping the derivations starting from ρ and $f(g(\sigma))$ we obtain $\rho \parallel f(g(\sigma)) \Longrightarrow \rho' \parallel f''(g''(\sigma')) \not\rightarrow$, hence we conclude $\rho' \xrightarrow{\mathsf{e}}$. □

It may be argued that $f \cdot g$ is somewhat "more powerful" than $f \circ g$, because $(f \circ g)(\sigma) \sqsubseteq (f \cdot g)(\sigma)$ but $(f \circ g)(\sigma) \not\simeq (f \cdot g)(\sigma)$ in general. Against this objection it is sufficient to observe that if f and g are k-orchestrators, then $f \cdot g$ is a $2k$-orchestrator. Thus, $f \cdot g$ is really nothing more than some proper combination of f and g, as expected.

4 Contract duality with orchestration

We tackle the problem of finding the dual contract $\rho^\perp$ of a given client contract ρ. Recall that $\rho^\perp$ should be the smallest (according to $\preceq$) contract such that ρ is compliant with $\rho^\perp$.

Before proceeding, we must face the fact that some clients cannot by satisfied by any service. For instance, there is no service that satisfies (the client) $\mathtt{0}$; similarly, there is no service that satisfies $\overline{a}.(\mathtt{0} \oplus b.\mathtt{e})$ since this client, after sending $\overline{a}$, may internally evolve into the state $\mathtt{0}$. We thus need a characterization of those (client) contracts that can be satisfied:

Definition 4.1 (viable contract)**.** A (client) contract ρ is *viable*, notation $\mathtt{viable}(\rho)$, if there exists σ such that $\rho \dashv \sigma$.

It is quite easy to provide an alternative, coinductive characterization of viable contracts.

Definition 4.2 (coinductive viability)**.** We say that the predicate $\mathscr{V}$ is a *coinductive viability* if $\rho \in \mathscr{V}$ and $\rho \Downarrow \mathrm{R}$ implies either $\mathtt{e} \in \mathrm{R}$ or $\rho(\alpha) \in \mathscr{V}$ for some $\alpha \in \mathrm{R}$.

This characterization mandates that no viable client contract can expose an empty ready set: every ready set must contain either the special action $\mathtt{e}$ denoting the client's ability to terminate successfully, or *at least* one action α whose continuation is itself a viable contract. So $\mathtt{e}$ is the simplest viable client contract, whereas $(a + b.\mathtt{e}) \oplus a.\overline{c}$ is not viable because its continuation after a is $\mathtt{0} \oplus \overline{c}$ that has an empty ready set.

Proposition 4.3. $\mathtt{viable}(\cdot)$ *is the largest coinductive viability.*

Proof. First we prove that $\mathtt{viable}(\cdot)$ is a coinductive viability. Let $\mathtt{viable}(\rho)$ and $\rho \Downarrow \mathrm{R}$. Then there exists σ such that $\rho \dashv \sigma$ and ρ' such that $\rho \Longrightarrow \rho' \not\longrightarrow$ and $\mathrm{R} = \mathtt{init}(\rho')$. If $\rho' \xrightarrow{\mathtt{e}}$ there is nothing to prove, so assume $\rho' \not\xrightarrow{\mathtt{e}}$ and $\sigma \Longrightarrow \sigma' \not\longrightarrow$. We have $\rho \parallel \sigma \Longrightarrow \rho' \parallel \sigma'$ and from $\rho \dashv \sigma$ we deduce that $\rho' \parallel \sigma' \longrightarrow \rho'' \parallel \sigma''$ for some ρ'' and σ''. Hence there exists α such that $\rho \Longrightarrow \rho' \xrightarrow{\overline{\alpha}} \rho''$ and $\sigma \Longrightarrow \sigma' \xrightarrow{\alpha} \sigma''$. It is trivial to see that from $\rho \dashv \sigma$ and $\rho \overset{\overline{\alpha}}{\Longrightarrow}$ and $\sigma \xrightarrow{\alpha}$ we have $\rho(\overline{\alpha}) \dashv \sigma(\alpha)$, hence we conclude $\mathtt{viable}(\rho(\overline{\alpha}))$.

To show that $\mathtt{viable}(\cdot)$ is indeed the largest coinductive viability, we show that any coinductive viability is included in $\mathtt{viable}(\cdot)$. To do this, assume that $\rho \in \mathscr{V}$ for some coinductive viability $\mathscr{V}$. We must be able to find a service $S(\rho)$ such that $\rho \dashv S(\rho)$. We define $S(\rho)$ thus

$$S(\rho) \stackrel{\text{def}}{=} \sum_{\rho \Downarrow \mathrm{R}, \alpha \in \mathrm{R} \setminus \{\mathtt{e}\}, \rho(\alpha) \in \mathscr{V}} \overline{\alpha}.S(\rho(\alpha))$$

and we leave the easy proof that $\rho \dashv S(\rho)$ to the reader. □

Now that we have a notion of viability, we are ready to define the dual contract.

Definition 4.4 (dual contract)**.** Let ρ be a viable client contract. The *dual contract* of ρ, denoted by $\rho^\perp$, is defined as:

$$\rho^\perp \stackrel{\text{def}}{=} \sum_{\rho \Downarrow \mathrm{R}, \mathtt{e} \notin \mathrm{R}} \bigoplus_{\alpha \in \mathrm{R}, \mathtt{viable}(\rho(\alpha))} \overline{\alpha}.\rho(\alpha)^\perp$$

The idea of the dual operator is to consider every state R of the client in which the client cannot terminate successfully ($\mathtt{e} \notin \mathrm{R}$). For every such state the service must provide at least one way for the client to proceed, and the least service that guarantees this is given by the internal choice of all the co-actions in $\overline{\mathrm{R}}$ that have viable continuations (note that there must be at least one of such actions because the client is viable by hypothesis). A few examples of dual contracts follow:

- $(a.\mathtt{e})^\perp = (a.\mathtt{e} \oplus \mathtt{e})^\perp = \overline{a}$ (the service must provide $\overline{a}$);
- $(a.\mathtt{e} + \mathtt{e})^\perp = \mathtt{0}$ (the service need not provide anything because the client can terminate immediately);
- $(a.\mathtt{e} + b.\mathtt{e})^\perp = \overline{a} \oplus \overline{b}$ (the service can decide whether to provide $\overline{a}$ or $\overline{b}$);
- $(a.\mathtt{e} \oplus b.\mathtt{e})^\perp = \overline{a} + \overline{b}$ (the service must provide both $\overline{a}$ and $\overline{b}$);
- $(\mathtt{rec}\ x.a.x)^\perp \simeq \mathtt{rec}\ x.\overline{a}.x$ (the service must provide an infinite sequence of $\overline{a}$'s).

Theorem 4.5 (duality)**.** *Let ρ be a viable client contract. Then*

(1) $\rho \dashv \rho^\perp$*;*
(2) $\rho \dashv \sigma$ *implies* $\rho^\perp \preceq \sigma$*.*

Proof. As regards item (1), consider a derivation $\rho \parallel \rho^\perp \Longrightarrow \rho' \parallel \sigma \not\longrightarrow$ and assume by contradiction that $\rho' \xrightarrow{\mathtt{e}}$. By unzipping this derivation we obtain that there exists φ such that $\rho \stackrel{\varphi}{\Longrightarrow} \rho' \not\longrightarrow$ and $\rho^\perp \stackrel{\overline{\varphi}}{\Longrightarrow} \sigma \not\longrightarrow$. In particular, by definition of $\rho^\perp$ we can rewrite this latter derivation as $\rho^\perp \stackrel{\overline{\varphi}}{\Longrightarrow} \rho(\varphi)^\perp \Longrightarrow \sigma \not\longrightarrow$. From $\rho' \parallel \sigma \not\longrightarrow$ we deduce $\overline{\mathtt{init}(\rho')} \cap \mathtt{init}(\sigma) = \emptyset$. Let $\mathrm{R}_1, \ldots, \mathrm{R}_n$ be the ready sets of $\rho(\varphi)$ not containing $\mathtt{e}$ (there must be at least one since $\rho' \xrightarrow{\mathtt{e}}$). From the fact that ρ is viable and by definition of $\rho^\perp$ we know that every ready set of $\rho(\varphi)^\perp$ contains one co-action from every ready set of $\rho(\varphi)$ that does not contain $\mathtt{e}$ and whose continuation is viable. Hence, $\mathtt{init}(\sigma) = \{\overline{\alpha}_1, \ldots, \overline{\alpha}_n\}$ where $\alpha_i \in \mathrm{R}_i$ and $\rho(\varphi\alpha_i)$ is viable. From $\rho(\varphi) \Longrightarrow \rho' \not\longrightarrow$ we deduce that $\mathtt{init}(\rho') = \mathrm{R}_k$ for some $k \in \{1, \ldots, n\}$. Now $\rho' \xrightarrow{\alpha_k}$ and $\sigma \xrightarrow{\overline{\alpha}_k}$, which contradicts $\overline{\mathtt{init}(\rho')} \cap \mathtt{init}(\sigma) = \emptyset$.

As regards item (2), it is sufficient to prove that $\mathscr{W} \stackrel{\text{def}}{=} \{(\tilde{\emptyset}, \rho(\overline{\varphi})^\perp, \sigma(\varphi)) \mid \rho \stackrel{\overline{\varphi}}{\Longrightarrow}, \sigma \stackrel{\varphi}{\Longrightarrow}\}$ is a coinductive weak 0-subcontract relation, because $(\tilde{\emptyset}, \rho^\perp, \sigma) \in \mathscr{W}$. Let $(\tilde{\emptyset}, \rho', \sigma') \in \mathscr{W}$. Then there exists φ such that $\rho' = \rho(\overline{\varphi})^\perp$ and $\sigma' = \sigma(\varphi)$. Consider $\mathrm{A} \stackrel{\text{def}}{=} \{\langle \alpha, \overline{\alpha} \rangle \mid \rho' \stackrel{\overline{\alpha}}{\Longrightarrow}\}$ and observe that $\tilde{\emptyset} \vdash_0 \mathrm{A}$. As

$$\text{(A1)}\quad \frac{\begin{array}{c}\mathrm{A}_r = \{\langle\varphi,\overline{\varphi}'\rangle \mid \sigma \stackrel{\varphi}{\Longrightarrow}, \tau \stackrel{\varphi'}{\Longrightarrow}, \mathbb{B} \vdash_k \langle\varphi,\overline{\varphi}'\rangle\} \\ \mathrm{A} = \{\langle\varphi,\overline{\varphi}'\rangle \in \mathrm{A}_r \mid \Gamma \cup \{(\mathbb{B},\sigma,\tau) \mapsto x\}, \mathbb{B}\langle\varphi,\overline{\varphi}'\rangle \vdash_k f_{\langle\varphi,\overline{\varphi}'\rangle} : \sigma(\varphi) \preceq^{\mathtt{a}} \tau(\varphi')\} \\ \tau \Downarrow \mathrm{S} \Rightarrow (\exists \mathrm{R} : \sigma \Downarrow \mathrm{R} \wedge \mathrm{R} \subseteq \mathrm{A} \circ \mathrm{S}) \vee (\emptyset \bullet \mathrm{A}) \cap \overline{\mathrm{S}} \neq \emptyset \qquad x \text{ fresh}\end{array}}{\Gamma, \mathbb{B} \vdash_k \mathtt{rec}\ x. \bigvee_{\mu \in \mathrm{A}} \mu.f_\mu : \sigma \preceq^{\mathtt{a}} \tau}$$

$$\text{(A2)}\quad \frac{\Gamma(\mathbb{B},\sigma,\tau) = x}{\Gamma, \mathbb{B} \vdash_k x : \sigma \preceq^{\mathtt{a}} \tau}$$

Table 1
Algorithm for deciding $\preceq$.

regards condition (1) in Definition 2.2, let $\{\mathrm{R}_1, \ldots, \mathrm{R}_n\} = \{\mathrm{R} \mid \rho \Downarrow \mathrm{R}, \mathtt{e} \notin \mathrm{R}\}$ be the ready sets of $\rho(\overline{\varphi})$ not containing $\mathtt{e}$. From the hypothesis $\rho \dashv \sigma$ we derive $\rho(\overline{\varphi}) \dashv \sigma(\varphi)$, hence $\overline{\mathrm{R}_i} \cap \mathrm{S} \neq \emptyset$ for every $1 \leq i \leq n$. Namely, for every $1 \leq i \leq n$ there exists $\overline{\alpha}_i \in \overline{\mathrm{R}_i} \cap \mathrm{S}$. By definition of dual contract we have $\rho(\overline{\varphi})^\perp \Downarrow \{\overline{\alpha}_1, \ldots, \overline{\alpha}_n\}$. We conclude $\{\overline{\alpha}_1, \ldots, \overline{\alpha}_n\} \subseteq \mathrm{A} \circ \mathrm{S}$. As regards condition (2), assume $\sigma(\varphi) \stackrel{\alpha}{\Longrightarrow}$ and $\langle\alpha,\overline{\alpha}\rangle \in \mathrm{A}$. Then $\sigma \stackrel{\varphi\alpha}{\Longrightarrow}$ and $\rho(\overline{\varphi})^\perp \stackrel{\alpha}{\Longrightarrow}$ hence $\rho \stackrel{\overline{\varphi\alpha}}{\Longrightarrow}$. By definition of $\mathscr{W}$ we conclude that $(\tilde{\emptyset}, \rho(\overline{\varphi})^\perp(\alpha), \sigma(\varphi)(\alpha)) \in \mathscr{W}$ because $\rho(\overline{\varphi})^\perp(\alpha) = \rho(\overline{\varphi\alpha})^\perp$ and $\sigma(\varphi)(\alpha) = \sigma(\varphi\alpha)$. □

The assumption of using orchestrators is essential as far as duality is concerned: $(a.\mathtt{e} + \mathtt{e})^\perp = \mathtt{0}$ but $\mathtt{0}$ is *not* the smallest (according to $\sqsubseteq$) contract satisfying $a.\mathtt{e} + \mathtt{e}$. For example, $\mathtt{0} \oplus b \sqsubseteq \mathtt{0}$ and $a.\mathtt{e} + \mathtt{e} \dashv \mathtt{0} \oplus b$. On the contrary, $\mathtt{0}$ is the least element of $\preceq$ and it can be used in place of any service contract that exposes an empty ready set. A notion a duality without orchestrators can only be achieved if the subcontract relation being considered provides a least element. This is possible for $\sqsubseteq$ if we extend the theory with diverging processes, as done in [31], although the duality operator turns out to be much more involved.

5 Synthesizing orchestrators

In this section we devise an algorithm for computing the k-orchestrator witnessing $\sigma \preceq \tau$, provided there is one. Actually, we have already seen that there can be more than one orchestrator proving a relation $\sigma \preceq \tau$, so when devising the algorithm we need a criterion for choosing a particular orchestrator as the "right" one. We know that orchestrators are closed under union, namely if $f : \sigma \preceq \tau$ and $g : \sigma \preceq \tau$, then $f \vee g : \sigma \preceq \tau$. So we may naively attempt to define an algorithm that synthesizes the *largest* orchestrator, according to their trace semantics:

Definition 5.1 (orchestrator ordering)**.** We say that the orchestrator f is

smaller than g, notation $f \leqslant g$, if $[\![f]\!] \subseteq [\![g]\!]$.

This approach is not effective since the largest orchestrator proving $\sigma \preceq \tau$ involves an infinite number of different names, and thus is not representable as a proper orchestrator. The idea is that, by means of Proposition 3.4, we may restrict our interest to the subclass of the orchestrators that are relevant for $\sigma \preceq \tau$.

Definition 5.2 (best relevant orchestrator)**.** We say that f is the *best relevant k-orchestrator* such that $f : \sigma \preceq_k \tau$ if $g : \sigma \preceq_k \tau$ and g is a relevant k-orchestrator implies $g \leqslant f$.

The algorithm that synthesizes the best relevant orchestrator proving $\sigma \preceq \tau$, provided there is one, is defined inductively by the rules in Table 1. A judgment of the form $\Gamma, \mathbb{B} \vdash_k f : \sigma \preceq^{\mathtt{a}} \tau$ means that f is a k-orchestrator proving that $\sigma \preceq \tau$ when the buffer of the orchestrator is in state $\mathbb{B}$. The context Γ memoizes triples $(\mathbb{B}, \sigma, \tau)$ so as to guarantee termination. The k-buffer $\mathbb{B}$ keeps track of the past history of the orchestrator (which messages the orchestrator has accepted and not yet delivered). We write $f : \sigma \preceq^{\mathtt{a}}_k \tau$ if $\emptyset, \tilde{\emptyset} \vdash_k f : \sigma \preceq^{\mathtt{a}} \tau$.

Let us comment on the rules of the algorithm. Although rule (A1) looks formidable, it embeds the conditions in Definition 2.6 in a straightforward way. Recall that the purpose of the algorithm is to find the best relevant orchestrator f such that every client strongly compliant with σ is weakly compliant with τ when this service is orchestrated by f, assuming that the buffer of the orchestrator is $\mathbb{B}$. Since $\mathbb{B}$ is a k-buffer, the number of enabled asynchronous orchestration actions is finite: an action $\langle \overline{a}, \varepsilon \rangle$ is enabled only if $\mathbb{B}(\circ, \overline{a}) > 0$; an action $\langle a, \varepsilon \rangle$ is enabled only if the buffer has not reached its capacity, namely if $\mathbb{B}(\bullet, \overline{a}) < k$; symmetrically for asynchronous service actions. Also, it is pointless to consider any orchestration action that would not cause any synchronization to occur. Hence, the set A_r of relevant, enabled orchestration actions in the first premise of the rule is finite. Of all the actions in this set, the algorithm considers only those in some subset A such that the execution of any orchestration action in A does not lead to a deadlock later on during the interaction. This is guaranteed if for every $\langle \varphi, \overline{\varphi'} \rangle \in \mathrm{A}$ we are able to find an orchestrator $f_{\langle \varphi, \overline{\varphi'} \rangle}$ that proves $\tau(\varphi') \preceq \sigma(\varphi)$ (second premise of the rule). When checking the continuations, the memoization context Γ is augmented associating the triple $(\mathbb{B}, \sigma, \tau)$ with a fresh orchestrator variable x, and the buffer is updated to account for the orchestration action just occurred. If the set A is large enough so as to satisfy the third premise of the rule, which is exactly condition (1) of Definition 3.7, then σ and τ can be related. The orchestrator computed in the conclusion of rule (A1) offers the union of all the relevant, enabled orchestration actions μ, each one followed by the corresponding continuation f_μ.

Rule (A2) is used when the algorithm needs to check whether there exists f such that $\Gamma, \mathbb{B} \vdash_k f : \sigma \preceq^{\mathtt{a}} \tau$ and $(\mathbb{B}, \sigma, \tau) \in \mathtt{dom}(\Gamma)$. In this case $\Gamma(\mathbb{B}, \sigma, \tau)$ is a variable that represents the orchestrator that the algorithm has already determined for proving $\sigma \preceq \tau$.

The algorithm described above is correct and complete and it always terminates.

Theorem 5.3. *The following properties hold:*

(1) (termination) it is decidable to check whether there exists f such that $f : \sigma \preceq^{\mathtt{a}}_k \tau$;
(2) (correctness) $f : \sigma \preceq^{\mathtt{a}}_k \tau$ implies that f has rank k and $f : \sigma \preceq_k \tau$;
(3) (completeness) $f : \sigma \preceq_k \tau$ and f is relevant for $\sigma \preceq_k \tau$ implies $g : \sigma \preceq^{\mathtt{a}}_k \tau$ for some g such that $f \leqslant g$.

The proof of correctness requires a cut-elimination result established by the following Lemma.

Lemma 5.4. *If $\Gamma \cup \{(\mathbb{B}, \sigma, \tau) \mapsto x\}, \mathbb{B}' \vdash_k f' : \sigma' \preceq^{\mathtt{a}} \tau'$ and $\Gamma, \mathbb{B} \vdash_k \mathtt{rec}\ x.f : \sigma \preceq^{\mathtt{a}} \tau$, then $\Gamma, \mathbb{B}' \vdash_k f'\{\mathtt{rec}\ x.f/_x\} : \sigma' \preceq^{\mathtt{a}} \tau'$.*

Proof. First of all observe that $\Gamma \subseteq \Gamma'$ and $\Gamma, \mathbb{B} \vdash_k f : \sigma \preceq^{\mathtt{a}} \tau$ implies $\Gamma', \mathbb{B} \vdash_k f : \sigma \preceq^{\mathtt{a}} \tau$. Consider the proof tree of $\Gamma \cup \{(\mathbb{B}, \sigma, \tau) \mapsto x\}, \mathbb{B}' \vdash_k f' : \sigma' \preceq^{\mathtt{a}} \tau'$. Such proof tree will contain $P_1, \ldots, P_n$ subtrees whose conclusion is $\Gamma_i, \mathbb{B} \vdash_k x : \sigma \preceq^{\mathtt{a}} \tau$ resulting from the application of rule (A2). We have $\Gamma \subseteq \Gamma_i$ for every $1 \leq i \leq n$, hence we can replace each subtree P_i with an instance of the proof tree of $\Gamma, \mathbb{B} \vdash_k \mathtt{rec}\ x.f : \sigma \preceq^{\mathtt{a}} \tau$, where every context is appropriately updated with Γ_i. We obtain a proof tree for $\Gamma, \mathbb{B}' \vdash_k f'\{\mathtt{rec}\ x.f/_x\} : \sigma' \preceq^{\mathtt{a}} \tau'$. □

We conclude this section with the proof of Theorem 5.3.

Proof of Theorem 5.3. As regards item (1), it is sufficient to show that there is a finite number of triples $(\mathbb{B}, \sigma, \tau)$ that can be stored in the environment Γ.

As regards the σ and τ components of the triple, this reduces to showing that the set $\{(\sigma(\varphi), \tau(\varphi')) \mid \sigma \stackrel{\varphi}{\Longrightarrow}, \tau \stackrel{\varphi'}{\Longrightarrow}\}$ is always finite and this is a direct consequence of Proposition 2.4. As regards the $\mathbb{B}$ component, let n be the number of different names occurring in either σ or τ. The number of different configurations of a k-buffer can be in, while proving that $\sigma \preceq \tau$, is $2kn$. Indeed for every name occurring in either σ or τ there can be at most k messages waiting to be delivered to the client and k messages waiting to be delivered to the service.

As regards item (2), by a simple structural induction it is easy to establish that, given a derivation for $\emptyset, \mathbb{B} \vdash_k f : \sigma \preceq^{\mathtt{a}} \tau$ where $\mathbb{B}$ is a k-buffer, every buffer $\mathbb{B}'$ in every judgment occurring in the derivation is also a k-buffer. It is sufficient to show that $\mathscr{W} \stackrel{\text{def}}{=} \{(\mathbb{B}, \sigma, \tau) \mid \emptyset, \mathbb{B} \vdash_k f : \sigma \preceq^{\mathtt{a}} \tau\}$ is a coinductive weak k-subcontract relation. Let $(\mathbb{B}, \sigma, \tau) \in \mathscr{W}$. Then $\emptyset, \mathbb{B} \vdash_k f : \sigma \preceq^{\mathtt{a}} \tau$ is derivable and furthermore the last rule applied must have been (A1) because the context Γ is empty. Let $\mathrm{A} \stackrel{\text{def}}{=} \mathtt{init}(f)$ and observe that $\mathbb{B} \vdash_k \mathrm{A}$. As regards condition (1) in Definition 3.2, there is nothing to prove because it exactly coincides with the third premise in rule (A1). As regards condition (2), assume $\tau \stackrel{\varphi'}{\Longrightarrow}$ and $\langle \varphi, \overline{\varphi}' \rangle \in \mathrm{A}$. From the first premise of rule (A1) we derive $\sigma \stackrel{\varphi}{\Longrightarrow}$. From the second premise we know that $\{(\mathbb{B}, \sigma, \tau) \mapsto x)\}, \mathbb{B}\langle \varphi, \overline{\varphi}' \rangle \vdash_k f_{\langle \varphi, \overline{\varphi}' \rangle} : \sigma(\varphi) \preceq^{\mathtt{a}} \tau(\varphi')$ is derivable. Since $\emptyset, \mathbb{B} \vdash_k f : \sigma \preceq^{\mathtt{a}} \tau$ is derivable by hypothesis, by Lemma 5.4 we obtain that $\emptyset, \mathbb{B}\langle \varphi, \overline{\varphi}' \rangle \vdash_k f_{\langle \varphi, \overline{\varphi}' \rangle}\{f/x\}\sigma(\varphi) \preceq^{\mathtt{a}} \tau(\varphi')$ is also derivable. We conclude $(\mathbb{B}\langle \varphi, \overline{\varphi}' \rangle, \sigma(\varphi), \tau(\varphi')) \in \mathscr{W}$ by definition of $\mathscr{W}$.

As regards item (3), from $g : \sigma \preceq \tau$ we derive that

$$\mathscr{W}_k \stackrel{\text{def}}{=} \{(\tilde{\emptyset}\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle, \sigma(\varphi_1 \cdots \varphi_n), \tau(\varphi'_1 \cdots \varphi'_n)) \mid g \xmapsto{\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle}\}$$

is a weak k-subcontract relation. Note that since g is relevant, we have that $g \xmapsto{\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle}$ implies $\sigma \stackrel{\varphi_1 \cdots \varphi_n}{\Longrightarrow}$ and $\tau \stackrel{\varphi'_1 \cdots \varphi'_n}{\Longrightarrow}$. By Proposition 2.4 and the fact that there is a finite number of configurations for k-buffer with finite domain we observe that $\mathscr{W}_k$ is finite. We prove that, if $(\mathbb{B}, \sigma, \tau) \in \mathscr{W}_k$, then $\Gamma, \mathbb{B} \vdash_k f : \sigma \preceq^{\mathtt{a}} \tau$ is derivable by induction on $\mathscr{W}_k \setminus \mathtt{dom}(\Gamma)$. The statement of the theorem follows by letting $\Gamma = \emptyset$.

The base case is when $(\mathbb{B}, \sigma', \tau') \in \mathtt{dom}(\Gamma)$, in which case there exists x such that $\Gamma(\mathbb{B}, \sigma', \tau') = x$ and we conclude immediately by rule (A2). In the inductive case, assume $(\mathbb{B}, \sigma', \tau') \notin \mathtt{dom}(\Gamma)$. From $(\mathbb{B}, \sigma', \tau') \in \mathscr{W}_k$ and by definition of $\mathscr{W}_k$ we deduce that there exist $\varphi_1, \ldots, \varphi_n$, $\varphi'_1, \ldots, \varphi'_n$ such that $g \xmapsto{\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle} g'$ and $\mathbb{B} = \tilde{\emptyset}\langle \varphi_1, \overline{\varphi}'_1 \rangle \cdots \langle \varphi_n, \overline{\varphi}'_n \rangle$ and $\sigma' = \sigma(\varphi_1 \cdots \varphi_n)$ and $\tau' = \tau(\varphi'_1 \cdots \varphi'_n)$. Since g is relevant for $\sigma \preceq \tau$ and has rank k, we deduce that $\mathtt{init}(g') \subseteq \mathrm{A}_r$ in the first premise of rule (A1). Let $\langle \varphi, \overline{\varphi}' \rangle \in \mathtt{init}(g')$ and let x be a fresh orchestrator variable. By definition of coinductive weak k-subcontract relation and from the fact that g is relevant we know that $(\mathbb{B}\langle \varphi, \overline{\varphi}' \rangle, \sigma'(\varphi), \tau'(\varphi')) \in \mathscr{W}_k$. Let $\Gamma' \stackrel{\text{def}}{=} \Gamma \cup \{(\mathbb{B}\langle \varphi, \overline{\varphi}' \rangle, \sigma', \tau') \mapsto x\}$. Since $\Gamma \subsetneq \Gamma'$, by induction hypothesis we obtain that there exists $f_{\langle \varphi, \overline{\varphi}' \rangle}$ such that $\mathbb{B}\langle \varphi, \overline{\varphi}' \rangle \vdash_k f_{\langle \varphi, \overline{\varphi}' \rangle} : \sigma'(\varphi) \preceq^{\mathtt{a}} \tau'(\varphi')$. Hence $\mathtt{init}(g') \subseteq \mathrm{A}$ in the second premise of rule (A1). Since g proves $\sigma \preceq \tau$, we have that $\mathtt{init}(g')$ satisfies condition (1) of Definition 3.7, which coincides with the third premise of rule (A1). From $\mathtt{init}(g') \subseteq \mathrm{A}$ we deduce that A also satisfies the third premise of rule (A1). Hence we can apply rule (A1) and conclude $\Gamma, \mathbb{B} \vdash_k \mathtt{rec}\ x.\bigvee_{\mu \in \mathrm{A}} f_\mu : \sigma' \preceq^{\mathtt{a}} \tau'$. The fact that the algorithm computes the best relevant orchestrator proving $\sigma \preceq \tau$ is an immediate consequence of $\mathtt{init}(g') \subseteq \mathrm{A}$, as shown earlier. □

6 An example: orchestrated dining philosophers

Consider a variant of the problem of the dining philosophers in which a service provider hires two philosophers for sending philosophical thoughts to the clients that provide two forks. Each philosopher is modeled by the following contract:

$$P_i \stackrel{\text{def}}{=} \texttt{rec}\ x.\mathit{fork}_i.\mathit{fork}_i.\overline{\mathit{thought}}.\overline{\mathit{fork}}.\overline{\mathit{fork}}.x$$

where the fork_i actions model the philosopher's request of two forks, $\overline{\mathit{thought}}$ models the generation of a thought, and the $\overline{\mathit{fork}}$ actions model the fact that the philosophers return both forks after having generated a thought. We decorate fork_i actions with an index i for distinguishing fork requests coming from different philosophers. Also, we need some way for describing the contract of two philosophers running in parallel. To this aim we make use of a parallel composition operator over contracts so that $\sigma \mid \tau$ stands for the interleaving of all the actions in σ and τ. Assuming that σ and τ never synchronize with each other, which is the case in this example, the $\mid$ operator can be expressed using a simplified form of *expansion law* [26]:

$$\sigma \mid \tau \stackrel{\text{def}}{=} \bigoplus_{\sigma\Downarrow_{\mathrm{R}},\tau\Downarrow_{\mathrm{S}}} \Big(\sum_{\alpha\in\mathrm{R}} \alpha.(\sigma(\alpha) \mid \tau) + \sum_{\alpha\in\mathrm{S}} \alpha.(\sigma \mid \tau(\alpha)) \Big)$$

where once again the well-foundedness of this definition is a consequence of Proposition 2.4.

The client modeled by the contract

$$C \stackrel{\text{def}}{=} \texttt{rec}\ x.\ \sum_{i=1..2} \overline{\mathit{fork}}_i.\ \sum_{i=1..2} \overline{\mathit{fork}}_i.\mathit{thought}.\mathit{fork}.\mathit{fork}.x$$

expects to receive an unbound number of thoughts, without ever getting stuck. The problem of this sloppy client is that it does not care that the two forks it provides end up to the same philosopher and this may cause the system to deadlock. To see whether such client can be made compliant with $P_1 \mid P_2$ we compute its dual contract

$$C^{\perp} \simeq \texttt{rec}\ x.\ \bigoplus_{i=1..2} \mathit{fork}_i.\ \bigoplus_{i=1..2} \mathit{fork}_i.\overline{\mathit{thought}}.\overline{\mathit{fork}}.\overline{\mathit{fork}}.x$$

and then we check whether $C^{\perp} \preceq P_1 \mid P_2$ using the algorithm. If we consider the sequence of actions $\overline{\mathit{fork}}_1\overline{\mathit{fork}}_2$ we reduce to checking

$$\overline{\mathit{thought}}.\overline{\mathit{fork}}.\overline{\mathit{fork}}.C^{\perp} \preceq P_1(\mathit{fork}_1) \mid P_2(\mathit{fork}_2)$$

The contract $P_1(\mathit{fork}_1) \mid P_2(\mathit{fork}_2)$ has only the ready set $\{\mathit{fork}_1, \mathit{fork}_2\}$, while the residual of the client's dual contract has only the ready set $\{\overline{\mathit{thought}}\}$. A similar situation occurs when considering the sequence of actions $\overline{\mathit{fork}}_2\overline{\mathit{fork}}_1$.

There is no orchestration action that can let the algorithm make some progress from these states.

In a sense the algorithm finds out that the two forks sent by the client must be delivered to the same philosopher, and this is testified by the resulting orchestrator

$$f \stackrel{\text{def}}{=} \texttt{rec}\ x.\ \bigvee_{i=1..2} \langle fork_i, \overline{fork_i}\rangle.\langle fork_i, \overline{fork_i}\rangle.\langle thought, \overline{thought}\rangle.\\ \langle fork, \overline{fork}\rangle.\langle fork, \overline{fork}\rangle.x$$

Suppose now that the service provider is forced to update the service with two new philosophers who, according to their habit, produce their thoughts only after having returned the forks. Their behavior can be described by the contract

$$Q_i \stackrel{\text{def}}{=} \texttt{rec}\ x.fork_i.fork_i.\overline{fork}.\overline{fork}.\overline{thought}.x$$

The service provider may wonder whether the clients of the old service will still be satisfied by the new one. The problem can be formulated as checking whether $P_1 \mid P_2 \preceq Q_1 \mid Q_2$ and the interesting step is when the algorithm eventually checks $P_1(fork_1 fork_1) \mid P_2 \preceq Q_1(fork_1 fork_1) \mid Q_2$ (symmetrically for P_2 and the sequence of actions $fork_2 fork_2$). At this stage $P_1(fork_1 fork_1) \mid P_2$ has just the ready set $\{\overline{thought}, fork_2\}$, whereas the contract $Q_1(fork_1 fork_1) \mid Q_2$ has just the ready set $\{\overline{fork}, fork_2\}$. By accepting the two $\overline{fork}$ messages asynchronously we reduce to checking whether $P_1(fork_1 fork_1) | P_2 \preceq \overline{thought}.Q_1 | Q_2$, which holds by allowing the $\overline{thought}$ action to occur, followed by the asynchronous sending of the two buffered $\overline{fork}$ messages. Overall the relation is proved by the orchestrator

$$g \stackrel{\text{def}}{=} \texttt{rec}\ x.\ \bigvee_{i=1..2} \langle fork_i, \overline{fork_i}\rangle.\ \bigvee_{i=1..2} \langle fork_i, \overline{fork_i}\rangle.\\ \langle \varepsilon, fork\rangle.\langle \varepsilon, fork\rangle.\langle thought, \overline{thought}\rangle.\langle \overline{fork}, \varepsilon\rangle.\langle \overline{fork}, \varepsilon\rangle.x$$

and now the sloppy C client will be satisfied by the service $Q_1 \mid Q_2$ by means of the orchestrator

$$f \cdot g = \texttt{rec}\ x.\ \bigvee_{i=1..2} \langle fork_i, \overline{fork_i}\rangle.\langle fork_i, \overline{fork_i}\rangle.\langle \varepsilon, fork\rangle.\langle \varepsilon, fork\rangle.\\ \langle thought, \overline{thought}\rangle.\langle \overline{fork}, \varepsilon\rangle.x$$

7 On the implementation of simple orchestrators

Simple orchestrators represent little more than just a *strategy* for satisfying the client: they constraint the actions of client and service so that interaction

is safe, or they prescribe messages that must be buffered and possibly delivered at a later stage of the interaction. Moreover, they are oblivious to the internal choices of client and service, so that the constraints they pose can only affect external choices. The consequence is that an orchestration-aware client or service can *internally* implement in a very easy way a simple orchestrator by following the corresponding strategy, which amounts at *filtering out* some actions (among those that are available) and *buffering* some messages, without otherwise interfering with the internal decisions of the process.

In this section we investigate three easily identifiable subclasses of simple orchestrators that can also be efficiently implemented *externally*, namely without any awareness from the client and from the service that any actual orchestration is being carried on. We start by defining the subclasses of *synchronous* and *asynchronous* orchestrators. As the name suggests, the former ones are simple orchestrators exclusively made of synchronous orchestration actions, whereas the latter ones are simple orchestrators exclusively made of asynchronous orchestration actions.

7.1 Virtual synchronous orchestrators

Let $\mathscr{I}(\sigma)$ be the canonical synchronous orchestrator for σ defined thus:

$$\mathscr{I}(\sigma) \stackrel{\text{def}}{=} \bigvee_{\sigma \stackrel{\alpha}{\Longrightarrow}} \langle \alpha, \overline{\alpha} \rangle . \mathscr{I}(\sigma(\alpha))$$

Namely, $\mathscr{I}(\sigma)$ is the orchestrator made of synchronous actions corresponding to the traces of σ. We write $\overline{f}$ for the orchestrator which has the same structure as f, but all the actions occurring in f have been turned into the corresponding co-actions.

Assume $f : \sigma \preceq \tau$ for some synchronous orchestrator f and $\mathscr{I}(\tau) \leqslant f$. This means that the orchestrator f includes the whole set of traces of τ, modulo the fact that f is made of orchestration actions and τ of plain actions. Namely, no action of τ is ever hidden by the orchestrator. It is easy to verify that under these circumstances conditions (1) and (2) of Definition 3.7 (weak subcontract) respectively reduce to conditions (1) and (2) of Definition 2.6 (strong subcontract). In other words, if $f : \sigma \preceq \tau$ and $\mathscr{I}(\tau) \leqslant f$, then $\sigma \sqsubseteq \tau$, hence *every* client that is satisfied by σ is also satisfied by τ. In this sense f is a *virtual* orchestrator, in that it does not carry on any actual orchestration.

Now assume $f : \sigma \preceq \tau$ for some synchronous orchestrator f and $\rho \dashv \sigma$ and $\overline{\mathscr{I}(\rho)} \wedge \mathscr{I}(\tau) \leqslant f$. Then the orchestrator f includes all the traces on which ρ and τ can synchronize. In other words, any synchronization between ρ and τ occurs also between ρ and σ and, once again, f never has to actually

hide any action from τ when the client is ρ. So, the particular client ρ is strongly compliant with τ ($\rho \dashv \tau$). For example, we have $\langle a, \overline{a}\rangle : a \preceq a + b$ and $\overline{a}.\mathsf{e} \dashv a$. Since $\overline{\mathscr{I}(\overline{a}.\mathsf{e})} = \langle a, \overline{a}\rangle.\langle \overline{\mathsf{e}}, \mathsf{e}\rangle$ and $\mathscr{I}(a + b) = \langle a, \overline{a}\rangle \vee \langle b, \overline{b}\rangle$ and $\langle a, \overline{a}\rangle.\langle \overline{\mathsf{e}}, \mathsf{e}\rangle \wedge (\langle a, \overline{a}\rangle \vee \langle b, \overline{b}\rangle) = \langle a, \overline{a}\rangle$, we may conclude $\overline{a}.\mathsf{e} \dashv a+b$. Note however that $a \not\sqsubseteq a + b$: for instance, the client $\rho \stackrel{\text{def}}{=} \overline{a}.\mathsf{e} + \overline{b}$ is such that $\rho \dashv a$ but $\rho \not\dashv a+b$. Indeed, $\overline{\mathscr{I}(\overline{a}.\mathsf{e} + \overline{b})} = \langle a, \overline{a}\rangle.\langle \overline{\mathsf{e}}, \mathsf{e}\rangle \vee \langle b, \overline{b}\rangle$ and $\overline{\mathscr{I}(\overline{a}.\mathsf{e} + \overline{b})} \wedge \mathscr{I}(a+b) = \mathscr{I}(a + b) \not\leqslant \langle a, \overline{a}\rangle$.

The subcontract relation of [31] is a special case of this kind of virtual orchestration, where the client never uses action names that were not included in the smaller contract.

7.2 Asynchronous orchestrators

According to the intuition behind orchestration, it should be possible to write a process that sits in between client and service and that somewhat implements the orchestrator. More formally, given a relation $f : \sigma \preceq \tau$ we should be able to find a CCS context C_f such that $C_f[\tau]$ is indistinguishable from (is a strong supercontract of) σ. The problem is that this context is not expressible in pure CCS, at least in the general case. To see why, consider the relation $\langle a, \overline{a}\rangle.\langle \overline{c}, c\rangle \vee \langle \overline{c}, c\rangle : a.\overline{c} \oplus \overline{c} \preceq a.\overline{c} \oplus \overline{c}$. In this example the orchestrator proves the reflexivity of $\preceq$ on the service contract $a.\overline{c} \oplus \overline{c}$. Observe that the given orchestrator is relevant and that any strictly smaller orchestrator does not suffice for proving the relation. We might try to implement the orchestrator in this particular case as the CCS context

$$C \stackrel{\text{def}}{=} (a.\overline{a}'.c'.\overline{c} + c'.\overline{c} \mid [\,][a'/a, c'/c]) \setminus \{a', c'\}$$

where as usual $[\,]$ denotes a hole in the context C which is meant to be replaced by (the service implementing) the contract $a \oplus \overline{c}$. The service is composed with a *forwarder* process $a.\overline{a'}.c'.\overline{c} + c'.\overline{c}$ whose only purpose is to pass messages from the client to the service and vice versa. To avoid confusion, the names a and c of the original service have been renamed to a' and c' respectively and the whole context has been restricted over these new names. Observe once again that the forwarder process does not try to add any new behavior, it simply interfaces client and service. The problem of this implementation is that the synchronization between client and service, which is *atomic* according to the interaction rules of Definition 2.1, has been encoded as two compound actions $a.\overline{a}'$ and $c'.\overline{c}$. This encoding enables computations that lead the client to a deadlock. For instance, if we consider the client $\rho \stackrel{\text{def}}{=} \overline{a}.c.\mathsf{e} + c.\mathsf{e}$, we have

$$\rho \parallel C[\tau] \longrightarrow c.\mathsf{e} \parallel (\overline{a}'.c'.\overline{c} \mid a'.\overline{c}' \oplus \overline{c}') \setminus \{a', c'\} \longrightarrow c.\mathsf{e} \parallel (\overline{a}'.c'.\overline{c} \mid \overline{c}') \setminus \{a', c'\} \not\longrightarrow$$

The problem lies in the fact that the synchronization semantics between client and service embedded in Definition 2.1 operates at the meta level of the calculus and cannot be modeled within the calculus itself. However, if we restrict the encoding to asynchronous orchestrators, everything works fine because client and service never synchronize directly with each other, but only indirectly with the mediation of the orchestrator.

More precisely, let $f : \sigma \preceq \tau$ and be f an asynchronous orchestrator. Then the context C_f can be defined as

$$C_f \stackrel{\text{def}}{=} (\mathscr{F}(f) \parallel [\,][a'/a \mid a \in \mathtt{names}(\tau)]) \setminus \{a' \mid a \in \mathtt{names}(\tau)\}$$

where $\mathscr{F}(f)$, the forwarder process corresponding to the orchestrator f, is defined as

$$\mathscr{F}(f) \stackrel{\text{def}}{=} \sum_{f \stackrel{\langle \alpha, \varepsilon \rangle}{\longmapsto} g} \alpha.\mathscr{F}(g) + \sum_{f \stackrel{\langle \varepsilon, \overline{\alpha} \rangle}{\longmapsto} g} \overline{\alpha}'.\mathscr{F}(g)$$

It can be shown that $\rho \dashv \sigma$ implies $\rho \dashv C_f[\tau]$, namely that $\sigma \sqsubseteq C_f[\tau]$.

As a concluding remark for this section, it should be noted that the algorithm presented in §5 can be trivially adapted so as to make it synthesize an asynchronous orchestrator proving a given relation, provided there is one. It is sufficient to restrict the set A in the first premise of rule (A1) (see Table 1) to asynchronous orchestration actions. In case a synchronous orchestrator is required, it is sufficient to run the algorithm with $k = 0$.

8 Related work

This work originated by revisiting CCS without τ's [18] in the context of Web services. Contracts are in fact just a concrete representation of *acceptance trees* [25, 26]. Early attempts to define a reasonable subcontract relation [10] have eventually led to the conclusion that some control over actions is necessary: [31] proposes a static form of control that makes use of explicit contract interfaces whereas [11] proposes a dynamic form of control by means of so-called *filters*. The present work elaborates on the idea of [11] by adding asynchrony and buffering to filters: this apparently simple addition significantly increases the technicalities of the resulting theory, both because of the very nature of asynchrony and also because orchestrator composition and conjunction no longer coincide. The subcontract relation presented in this work, because of its liveness-preserving property, has connections with and extends the subtyping relation on session types [27, 23] and stuck-free conformance relation [22]. A more detailed comparison with these works can be found in [11]. [21] provides a very clear and interesting comparison of several refinement relations among which *reduction* refinement, which corresponds to the *must* preorder

and to our notion of strong subcontract (see §2), and *implementation* refinement, which roughly corresponds to the subcontract relation defined in [10] and that can be traced back to the LOTOS language [9]. The authors of [21] emphasize the importance of implementation refinement, which enables *width* and *depth* extensions of partial specifications, but also its lack of transitivity, which hinders its application in practice. The present work (as well as [11]) can be seen as a solution to the lack of transitivity of this (and similar) refinement relations, by the introduction of suitable coercions/orchestrators/connectors.

A closely related work regarding contracts for Web services is [8]. The main conceptual difference between [8] and the present work regards the destination usage of the contract language: [8] defines a subcontract relation for reasoning on the modular refinement of Web services within coreographies, whereas here we focus on Web service discovery. As a consequence, the subcontract relation defined in [8] is a precongruence with respect to the parallel composition operator but it is stricter than $\preceq$ in the present work. In this sense it is closer to the strong subcontract relation in this work (§2), and is actually a fair, partially asynchronous variant of it. From a technical point of view there are several differences: in [8] the contract language is basically a full process calculus, whereas we focus on just two basic choice operators since we only care about the external, observable behavior of a Web service and not of its implementation; also, repeated behavior is modeled in [8] using the Kleene star operator, while we use recursion. In a nondeterministic setting it is well known that the Kleene star operator is unable to capture some behaviors that are expressible by means of recursion [19, 2], but it is unclear to which extent this limitation of the Kleene operator makes a difference in practice. Finally, [8] adopts a less abstract communication model based on *partial asynchrony*, whereby output actions cannot be negotiated by the receiver, but they do have a continuation. This allows the subcontract relation to enjoy width extensions of input actions without the intervention of any orchestrator, if the context never provides corresponding output actions. We have seen in §7 that this feature is subsumed by our subcontract relation, whereby width extensions are safe (without orchestration) for those clients that never try to synchronize on the new actions of the extended contract.

WS-BPEL [1] is often presented as an orchestration language for Web services. Remarkably WS-BPEL features boil down to storing incoming messages into variables (buffering) and controlling the interactions of other parties. Our orchestrators can be seen as streamlined WS-BPEL orchestrators in which all the internal nondeterminism of the orchestrator itself is abstracted away. ORC [33] is perhaps the most notable example of orchestration-oriented, algebraic language. The peculiar operators $\gg$ and **where** of ORC represent different forms of *pipelining* and can be seen as orchestration actions in conjunction with the composition operator $\cdot$ of simple orchestrators (§3).

There has been extensive research on the automatic synthesis of connectors both in the domain of software architectures (see for example [29]) and also in the more specific domain of Web services [37, 5, 28, 35, 24]. In these contexts the problem consists in finding a connector component (if there is one) which coordinates n given components (associated with corresponding behaviors) so as to accomplish a specific goal (for example, adhering to a target behavior). There is a clear analogy with the present work in that a component of the system (which we call orchestrator) is synthesized so as to make other components interact in some restricted way. We can highlight four main differences between the two scenarios: (1) there is a conceptual difference in that we focus on *finding* an existing service with a desired behavior, whereas Web service composition tries to synthesize a desired behavior starting from n given services. (2) The nature of simple orchestrators is driven by a notion of safe replacement for Web services (the subcontract relation). Although they play an essential role, simple orchestrators are just a tool for reasoning on service equivalence, which is the real main concern in this work. (3) The connector resulting from the automatic composition of Web services is *ad hoc* for the particular set of services that have been composed. In our case, it is possible to synthesize a *universal orchestrator* that satisfies all the clients of a desired service. The tight relationship between the subcontract relation and orchestrators provides us with an efficient way of composing connectors, which is not possible in the more complex scenario of Web service composition. (4) The automatic composition of Web services can only generate stub connectors whose low-level details must still be filled in by programmers. This is due to the fact that in most cases the behavioral description of the Web services is not detailed enough (or is too complex) to fully automate the code generating process. In our restricted scenario, the orchestrators are simple enough to admit a fully automatic code generation.

The present paper corrects and improves [34] in several ways. The main changes, other than the presence of the proofs for all the results stated in the paper and a more thorough comparison with related work, are the following: first and foremost we use direct characterizations of compliance where [34] used coinductive ones. The current presentation provides a better intuition behind the notions of compliance, especially for orchestrated systems, and results in the simplification of some proofs. The downside is that the orchestration language must be introduced since the very beginning of §3, whereas in [34] it emerges from the very notion of (weak) compliance. Second, we generalize the dual operator to so-called *viable* contracts. Third, we present a deterministic variant of the synthesis algorithm that is proved to generate the *best* orchestrator for a given relation, in the same spirit as has been done in [11]. Fourth, we discuss some important subclasses of orchestrators which admit efficient implementations and thus we show how the presented algorithm can be easily modified for running specialized queries.

9 Discussion

We can identify two main contributions of this work. On the theoretical side, we have adapted the testing framework [17, 26] by assuming that orchestrators can mediate the interaction between a client and a service. Clients are tests and services are processes to be tested. The notion of passing a test is captured by the compliance relations: strong compliance denotes successful interaction without the help of an orchestrator, whereas weak compliance denotes successful interaction with the help of an orchestrator. This approach solves the lack of transitivity in existing refinement relations [21] by means of simple adapters that we call orchestrators. Orchestrators can be interpreted as morphisms (or explicit behavioral coercions) transforming service contracts (§3.4). The morphism induced by an orchestrator enjoys nice and useful properties and tightly links the strong and weak compliance relations. We have provided different characterizations for the subcontract relations: the semantic, set-theoretic ones are based on the corresponding compliance relation, whereas the coinductive ones are more amenable for investigation and implementation. While coinductive characterizations of the strong subcontract relation and proper subsets of the weak subcontract relation are known in the literature (see [31, 11] but also [14]), the coinductive characterization for a testing relation involving asynchrony is original to the best of the author's knowledge.

On the practical side, we have defined a decidable, liveness-preserving subcontract relation for discovering Web services by means of their contract. The subcontract relation is coarser than and subsumes the existing ones, thus enlarging the set of services satisfying a given client and favoring service reuse. Orchestrators are pivotal in this respect. The existence of a universal orchestrator that is independent of the client allows us to precompute the orchestrator proving a given relation between service contracts and to cache it in the Web service registry. The algorithm for $\preceq$ can be easily specialized for deciding relations that are stricter than $\preceq$ and that permit efficient orchestrator implementations. While we have focused on Web services discovery by means of contracts, the subcontract relation as it stands can also be proficiently used in different contexts, such as Web service static verification, implementation and refactoring, where a notion of Web service equivalence is required.

The synthesis algorithm is computationally expensive. It is well known that deciding $\sqsubseteq$ is PSPACE-complete [30] although common practice suggests that worst cases occur seldom [14]. In our setting more complexity is added for synthesizing the orchestrator, although it is not evident to which extent this makes a difference in practice. It is also unclear whether it is possible to enhance the running cost of the algorithm by means of heuristics or other programming techniques, since previous work dealing with connector synthesis [5, 24] seems

to suggest that the problem is intrinsically hard. Furthermore, the algorithm is parameterized by an index k that roughly indicates the amount of buffering that is permitted. This index might be set to a constant (specified or determined by the entity that is in charge of the orchestration) for limiting orchestration costs, or it may be an argument of the query sent by the client. While it is reasonable to expect that relatively small values for k are sufficient for significantly improving the flexibility given by $\preceq$, it would also be interesting to be able to compute, if it exists, the orchestrator with lowest rank that proves a given relation $\sigma \preceq \tau$. Intuitively it should be possible to determine an upper bound using the regularity of σ and τ, although a precise result is still lacking.

Asynchronous variants of the classical testing preorders [12, 7] are notoriously more involved than their synchronous counterparts and they are usually defined assuming that self-synchronization is possible and that output messages are allowed to float around in unbounded buffers. Since these assumptions do not reflect the practice of Web services, our development can be seen as a practical variant of the classical asynchronous testing theories. In particular, it might be interesting to try to reduce the asynchronous *must* preorder without self-synchronization to our subcontract relation by analyzing the structure of orchestrators proving the relation (an orchestrator that always enables all of its asynchronous input and output actions acts like an unbounded buffer).

References

[1] Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, et al. *Web Services Business Process Execution Language Version 2.0*, 2007.

[2] Jos C. M. Baeten, Flavio Corradini, and Clemens A. Grabmayer. A characterization of regular expressions under bisimulation. *J. ACM*, 54(2):6, 2007.

[3] Arindam Banerji, Claudio Bartolini, Dorothea Beringer, Venkatesh Chopella, et al. *Web Services Conversation Language (*WSCL*) 1.0*, 2002.

[4] Tom Bellwood, Steve Capell, Luc Clement, John Colgrave, et al. UDDI *Version 3.0.2*, 2005. OASIS Standard, `http://uddi.org/pubs/uddi-v3.0.2-20041019.htm`.

[5] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Automatic composition of e-services that export their behavior. In *In Proc. 1st Int. Conf. on Service Oriented Computing (ICSOC), volume 2910 of LNCS*, pages 43–58. Springer, 2003.

[6] Dorothea Beringer, Harumi Kuno, and Mike Lemon. *Using* WSCL *in a* UDDI *Registry 1.0*, 2001.

[7] Michele Boreale, Rocco De Nicola, and Rosario Pugliese. Trace and test-

ing equivalence on asynchronous processes. *Information and Computation*, 172(2):139–164, 2002.

[8] Mario Bravetti and Gianluigi Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *Proc. of the 6th Intl. Symposium on Software Composition*. Springer, 2007.

[9] Ed Brinksma, Giuseppe Scollo, and Chris Steenbergen. LOTOS *specifications, their implementations and their tests*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.

[10] Samuele Carpineti, Giuseppe Castagna, Cosimo Laneve, and Luca Padovani. A formal account of contracts for Web Services. In *WS-FM'06*, number 4184 in LNCS, pages 148–162. Springer, 2006.

[11] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for Web services. In *Proceedings of POPL'08*, pages 261–272. ACM, 2008.

[12] Ilaria Castellani and Matthew Hennessy. Testing theories for asynchronous languages. In *Proceedings of Foundations of Software Technology and Theoretical Computer Science*, number 1530 in LNCS, pages 90–101. Springer, 1998.

[13] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. *Web Services Description Language (*WSDL*) Version 2.0 Part 1: Core Language*, 2007.

[14] Rance Cleaveland and Matthew Hennessy. Testing equivalence as a bisimulation equivalence. *Formal Aspects of Computing*, 5(1):1–20, 1993.

[15] John Colgrave and Karsten Januszewski. Using WSDL in a UDDI registry, version 2.0.2. Technical note, OASIS, 2004.

[16] Bruno Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169, 1983.

[17] Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[18] Rocco De Nicola and Matthew Hennessy. CCS without τ's. In *TAPSOFT'87/CAAP'87*, number 249 in LNCS, pages 138–152. Springer, 1987.

[19] Rocco De Nicola and Anna Labella. Nondeterministic regular expressions as solutions of equational systems. *Theor. Comput. Sci.*, 302(1-3):179–189, 2003.

[20] Roberto Di Cosmo. *Isomorphisms of Types: from Lambda Calculus to Information Retrieval and Language Design*. Birkhauser, 1995.

[21] Rik Eshuis and Maarten M. Fokkinga. Comparing refinements for failure and bisimulation semantics. *Fundamenta Informaticae*, 52(4):297–321, 2002.

[22] Cédric Fournet, C. A. R. Hoare, Sriram K. Rajamani, and Jakob Rehof. Stuck-free conformance. Technical Report MSR-TR-2004-69, Microsoft Research, 2004.

[23] Simon Gay and Malcolm Hole. Subtyping for session types in the π-calculus. *Acta Informatica*, 42(2-3):191–225, 2005.

[24] Giuseppe De Giacomo and Sebastian Sardiña. Automatic synthesis of new behaviors from a library of available behaviors. In *IJCAI*, pages 1866–1871, 2007.

[25] Matthew Hennessy. Acceptance trees. *JACM: Journal of the ACM*, 32(4):896–928, 1985.

[26] Matthew Hennessy. *Algebraic Theory of Processes*. Foundation of Computing. MIT Press, 1988.

[27] Kohei Honda. Types for dyadic interaction. In *CONCUR'93*, number 715 in LNCS, pages 509–523, 1993.

[28] Richard Hull, Michael Benedikt, Vassilis Christophides, and Jianwen Su. E-services: a look behind the curtain. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–14, New York, NY, USA, 2003. ACM.

[29] Paola Inverardi and Massimo Tivoli. Software architecture for correct components assembly. In *SFM'03*, pages 92–121, 2003.

[30] Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

[31] Cosimo Laneve and Luca Padovani. The *must* preorder revisited – an algebraic theory for web services contracts. In *CONCUR'07*, number 4703 in LNCS, pages 212–225. Springer, 2007.

[32] Cosimo Laneve and Luca Padovani. The pairing of contracts and session types. In *Festschrift for Ugo Montanari on the Occasion of his 65th Birthday*, number 5065 in LNCS, pages 681–700. Springer, 2008.

[33] Jayadev Misra and William R. Cook. Computation orchestration – a basis for wide-area computing. *Software and Systems Modeling*, 6(1):83–0110, 2007.

[34] Luca Padovani. Contract-directed synthesis of simple orchestrators. In *CONCUR'08*, number 5201 in LNCS, pages 131–146. Springer, 2008.

[35] Marco Pistore, Paolo Traverso, Piergiorgio Bertoli, and Annapaola Marconi. Automated synthesis of composite BPEL4WS web services. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, pages 293–301, Washington, DC, USA, 2005. IEEE Computer Society.

[36] Mikael Rittri. Retrieving library functions by unifying types modulo linear isomorphism. *RAIRO Theoretical Informatics and Applications*, 27(6):523–540, 1993.

[37] Paolo Traverso and Marco Pistore. Automated composition of semantic web services into executable processes. In *International Semantic Web Conference*, pages 380–394, 2004.

[38] Claus von Riegen and Ivana Trickovic. Using BPEL4WS in a UDDI registry. Technical note, OASIS, 2004.