

Contract-Based Discovery and Adaptation of Web Services

Luca Padovani

Istituto di Scienze e Tecnologie dell'Informazione – Università di Urbino
padovani@sti.uniurb.it

Abstract. A *contract* describes the observable behavior of a Web service. When looking for Web services providing specific capabilities, the contract can be used as an important search key. This calls for a notion of contract equivalence that goes beyond nominal or structural equivalence.

In this paper we define a simple, yet expressive formal language for describing Web service contracts. We provide a natural, set-theoretic semantics of contracts and we use it for defining a family of equivalence relations that can be effectively used for discovering and adapting Web services implementing a specific contract.

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 [12], and also the behavior – or *contract* – supported by the service [3, 1].

As an example, Figure 1 shows (a streamlined fragment of) the WS-BPEL document describing the behavior of a Web service that waits for purchase orders from a client, verifies that the client has enough credit for the purchase, and arranges the delivery of the ordered items with the deposit if the ordered item is available. The `receive` operation (line 3) waits for the order from the client, and stores the order information into a local variable `Request`. The `flow` activity (lines 4–9) invokes the credit and deposit services in parallel, and waits for the answer from both invocations. Then, the `switch` activity (lines 10–23) checks the responses from the credit and deposit services. If both responses are positive, namely if the client has enough credit and if the ordered item is available from the deposit (lines 11–12), the service sends a shipment notification to the deposit (line 13) and a confirmation to the client (line 14). If the item is not available, but the client does have enough credit (line 16), then the service issues a refund to the client (line 17) and notifies the client that the order was unsuccessful (line 18). In the remaining cases (lines 20–22) the service simply notifies the client of the unsuccessful transaction (line 21).

A service is advertised by registering its description in one or more Web service repositories [4, 13, 6, 31] that can be queried for discovering services satisfying a particular client. This calls for a formalization of the contract language and of a notion of client satisfaction.

```
1 <process>
2   <sequence>
3     <receive operation="Order" variable="Request"/>
4     <flow>
5       <invoke operation="Deposit" inputVariable="Request"
6         outputVariable="Deposit"/>
7       <invoke operation="Credit" inputVariable="Request"
8         outputVariable="Credit"/>
9     </flow>
10    <switch>
11      <case condition="getVariableData(Deposit) == true
12        && getVariableData(Credit) == true">
13        <invoke operation="Ship" inputVariable="Request"/>
14        <reply operation="Order" value="OK"/>
15      </case>
16      <case condition="getVariableData(Credit) == true">
17        <invoke operation="Refund" inputVariable="Request"/>
18        <reply operation="Order" value="NO"/>
19      </case>
20      <otherwise>
21        <reply operation="Order" value="NO"/>
22      </otherwise>
23    </switch>
24  </sequence>
25 </process>
```

Fig. 1. WS-BPEL description of ordering service.

The contract language. As regards the formalization of the contract language, we will focus on the *observable behavior* of services by abstracting every detail that is specific to a particular implementation. For example, by looking at Figure 1, we realize that one possible abstraction is given by the actions O, D, C, S, and R (which we use for the sake of brevity in place of the more verbose Order, Deposit, Credit, Ship, and Refund in Figure 1) that are performed by the service. The fact, for example, that the service declares a local variable Request is not interesting, as far as the observable behavior is concerned. However, this abstraction of the service behavior is too weak, because it does not say anything about the *order* in which those actions are (or can be) executed by the service. A more precise abstraction of the service behavior is to consider the set of *traces* of actions that can be performed by the service. In the example of Figure 1 we have 6 possible traces, namely

$$\{\text{ODCSO}, \text{OCDSO}, \text{ODCRO}, \text{OCDR0}, \text{ODCO}, \text{OCDO}\}$$

which are obtained by considering the 3 outcomes of the transaction (successful, unsuccessful because the item is not available, all the remaining cases) times the possible interleaving of the actions D and C in the flow activity (lines 5–10). In practice, we are approximating the behavior of two activities occurring in parallel with all the possible permutations of these actions (interleaving semantics).

This is without doubt a more precise description of the service, but still it has a major shortcoming in that it confuses two very different choices. On one hand, the simultaneous presence of traces of the form ODs and traces of the form OCs must be interpreted as the fact that the service does not mandate which of the two actions D and C should occur first. More technically, this is an *external choice* between the actions D and C. It is the environment in which the service operates, and not the service itself, that decides which of the two actions to perform. On the other hand, the simultaneous presence of traces such as sSO, sRO, and sO must be interpreted as the fact that the service decides whether the order is successful, in which case an S action is performed followed by the notification to the client, or unsuccessful because the item is unavailable, in which case a R action is performed followed by the notification to the client, or only a notification is sent. More technically, this is an *internal choice* between the actions S, R, and O, each one followed by a corresponding continuation. It is the service that autonomously decides, depending on its internal state, which of these actions to perform.

In summary, the contract of a service is a structured term that describes the actions performed, the order in which these actions are performed, and the branching points (corresponding to external and internal choices) in the service behavior. Additionally, we will distinguish actions denoting *incoming messages* received by the service, from actions denoting *outgoing messages* that correspond to invocations to other services or to responses to the client of the service. This distinction will be essential for determining possible causal dependencies between actions, as we will see in a later section. More technically, we express contracts using a fragment of CCS [16] with two choice operators (+ for external choice and \oplus for internal choice) without relabeling, restriction, and parallel composition. For example, the service in Figure 1 can be described by the term

$$O.(\bar{D}.C.D.C.(\bar{S}.\bar{O} \oplus \bar{R}.\bar{O} \oplus \bar{O})) + \bar{C}.\bar{D}.D.C.(\bar{S}.\bar{O} \oplus \bar{R}.\bar{O} \oplus \bar{O}))$$

where we have distinguished the requests \bar{D} and \bar{C} from the corresponding responses D and C .¹

The subcontract relation. As regards the formalization of the satisfaction relation between a client and a service, the intuition is that a client is *compliant with* (or *satisfied by*) a service if every possible interaction between the client and the service leads the client into a successful state. If we represent the behavior of the client by means of a contract ρ and σ is the contract of the service, we denote this fact by writing $\rho \dashv \sigma$. The definition of compliance calls for two more notions: that of interaction between client and service, and that of successful state for a client. We will postpone the exact definition of these two notions to a later section. For the time being, we observe that a formal notions of compliance may be used for implementing contract-based query engines. The query for services that satisfy ρ is answered with the set $\mathcal{Q}(\rho) = \{\sigma \mid \rho \dashv \sigma\}$.

A major drawback of this approach is that the complexity of running a query grows with the number of services registered in the repository, independently of the fact that many registered services may actually be equivalent, in terms of the clients they satisfy. In fact, it makes sense to relax this equivalence relation into a *subcontract relation*: we say that σ is a subcontract of τ , notation $\sigma \preceq \tau$, if every client satisfied by σ is also satisfied by τ . In this sense, any service with contract τ (or greater, according to \preceq) can appear in the answer of a query for a service σ if $\sigma \preceq \tau$. Now, if we are able to compute the *canonical service* that satisfies ρ , namely the smallest (according to \preceq) contract ρ^\perp such that $\rho \dashv \rho^\perp$, then we can answer the above query with the set $\mathcal{Q}(\rho) = \{\sigma \mid \rho^\perp \preceq \sigma\}$. The advantage of this approach is that the \preceq relation between services can be precomputed as 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.

Technical issues aside, it is possible to argue about some of the properties that we expect from the subcontract relation. For example, it is reasonable to expect that $\sigma \oplus \tau \preceq \sigma$, namely that it is possible to use a service with contract σ in place of a service that internally decides to behave according to either σ or τ . The larger service is more deterministic than the smaller one. In general, we expect \preceq to *reduce nondeterminism*. In a dual manner, it is also reasonable to expect that $\sigma \preceq \sigma + \tau$, namely that it is possible to use a service that externally offers more possible behaviors in place of a service that offers a subset of them. Unfortunately, this relation is much more subtle, and it does not hold in general because the additional behavior τ may cause interferences with σ and also with clients of the smaller service. We will see that it is possible to partially guarantee this desirable property if we assume that the interaction between the client and a service is mediated by a suitable orchestrator. In a context where a third process, the orchestrator, helps client and service to interact smoothly, other relations become

¹ We have deliberately chosen to read the response from the deposit service first, followed by the response from the credit service. This is not restrictive since the service cannot proceed until both responses are received.

feasible. For example, it would be reasonable to expect that $a.b.\sigma \preceq b.a.\sigma$, namely that the order in which subsequent input actions are performed by the service should be irrelevant and similarly for sequences of output actions. In general, it should be possible to replace a service with contract $a.\bar{b}.\sigma$ with another one with contract $\bar{b}.a.\sigma$, since the latter is able to send the b message without needing an a message from the client. Conversely, it should not be possible to replace a service with contract $\bar{a}.b.\sigma$ with another one with contract $b.\bar{a}.\sigma$, since a client of the first service may need the content of the a message before being able to send b to the service.

The aim of this tutorial is the definition of a subcontract relation that can be effectively 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 as 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.

Structure of the paper. In §2 we define syntax and semantics of the contract language and we define strong variants of the compliance and subcontract relations. We will see that these relations enjoy nice properties, but are too strict for the purposes of Web service discovery. 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 that blocks some actions and permits others. We proceed by studying simple orchestrators and the fundamental properties of the weak relations they induce, including their connection with the corresponding strong variants and a sound and complete deduction system for the weak subcontract relation. In §4 we give orchestrators the ability of buffering messages from the client or from the service, and of delivering them at later stages in the interaction. We will go through a similar round of properties and results as we did for the simple orchestrators in §3. §5 shows how to compute the principal dual contract ρ^\perp of a client contract ρ . In §6 we argue that all the definitions and results from previous sections can be naturally extended to the case of potentially infinite behaviors. We will do so by departing from standard process-algebraic techniques and by adopting a more basic, yet elegant approach. In §7 we devise an algorithm for the subcontract relation defined in §4, which includes the ones in §2 and in §3. The algorithm is proved to be sound and complete. §8 provides a somewhat more extensive example of application of the theory to an adaptation of the dining philosophers problem in the Web service setting. §9 concludes the paper with a survey of some closely related work and a sketch of possible tracks of future research. Long proofs and auxiliary results have been moved to the appendix for improving readability.

2 A theory of contracts

2.1 Contracts: syntax and semantics

Let us begin by fixing some notation. The syntax of contracts makes use of a denumerable set \mathcal{N} of *names* ranged over by a, b, \dots ; we write $\bar{\mathcal{N}}$ for the set of *co-names* \bar{a} ,

where $a \in \mathcal{N}$. Names represent input actions, while co-names represent output actions; we let α, β, \dots range over actions, namely elements of $\mathcal{N} \cup \overline{\mathcal{N}}$; we let φ, φ', \dots range over strings of actions, ε being the empty string as usual; we let R, S, \dots range over finite sets of actions; we let $\overline{\alpha} = \alpha$ and $\overline{R} = \{\overline{\alpha} \mid \alpha \in R\}$ 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.

Definition 1 (contract syntax). *Contracts are ranged over by $\rho, \sigma, \tau, \dots$ and their syntax is given by the following grammar:*

$\sigma ::=$	contract	$\alpha ::=$	action
	0		a
	$\alpha.\sigma$		\overline{a}
	$\sigma + \sigma$		
	$\sigma \oplus \sigma$		

The null contract 0 describes the idle process that offers no action; 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 . For the sake of brevity we will omit trailing 0 's, so for instance we will write $a.b$ in place of $a.b.0$.

We proceed by giving contracts an operational semantics that describes how they evolve over time and which actions they offer. In this chapter, we will blur the distinction between processes and contracts so that whenever we speak of a contract that emits or offers an action or exhibits a certain behavior, what we really mean is that the process respecting the contract emits or offers the same action or exhibits the same behavior.

Definition 2 (operational semantics of contracts). *The operational semantics of contracts is described by the rules below*

$$\alpha.\sigma \xrightarrow{\alpha} \sigma \quad \sigma \oplus \tau \longrightarrow \sigma \quad \frac{\sigma \xrightarrow{\alpha} \sigma'}{\sigma + \tau \xrightarrow{\alpha} \sigma'} \quad \frac{\sigma \longrightarrow \sigma'}{\sigma + \tau \longrightarrow \sigma' + \tau}$$

plus the symmetric of the last three rules.

The relation \longrightarrow denotes internal, invisible transitions, while $\xrightarrow{\alpha}$ denotes visible transitions labeled with an action α . The first rule states that a contract $\alpha.\sigma$ may offer an action α and evolve to the residual contract σ . The second rule states that a contract $\sigma \oplus \tau$ may evolve to σ (or to τ) by means of an invisible, internal transition. The third rule states that a contract $\sigma + \tau$ offers all the actions that are offered by either σ or τ . Finally, the last rule states that $+$ is a truly external choice: the internal transition $\sigma \longrightarrow \sigma'$ does not preempt the τ branch.² We write \Longrightarrow for the reflexive, transitive

² The transition relation of contracts is the same as that of CCS without τ 's [16].

closure of \longrightarrow ; let \Longrightarrow be $\Longrightarrow \xrightarrow{\alpha} \Longrightarrow$; we write $\sigma \xrightarrow{\alpha}$ if there exists σ' such that $\sigma \xrightarrow{\alpha} \sigma'$, and similarly for $\sigma \xrightarrow{\bar{\alpha}}$; let $\text{init}(\sigma) \stackrel{\text{def}}{=} \{\alpha \mid \sigma \xrightarrow{\alpha}\}$.

We now have all the technical instruments for defining how a client and a service interact and what it means for a client to be in a successful state. As regards the latter notion, we reserve a special action e (for “end”) that we assume to occur in client contracts only and we say that a client ρ is satisfied if $\rho \xrightarrow{e}$, namely if it has the immediate possibility of offering an e action. Observe that, by the last rule in Definition 2, if $\rho \xrightarrow{e}$, then $\rho' \xrightarrow{e}$ for every residual ρ' such that $\rho \Longrightarrow \rho'$. In general, once an action is offered *externally* by a contract, it cannot be revoked by means of internal moves of the same contract.

Definition 3 (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:

$$\frac{\rho \longrightarrow \rho'}{\rho \parallel \sigma \longrightarrow \rho' \parallel \sigma} \quad \frac{\sigma \longrightarrow \sigma'}{\rho \parallel \sigma \longrightarrow \rho \parallel \sigma'} \quad \frac{\rho \xrightarrow{\bar{\alpha}} \rho' \quad \sigma \xrightarrow{\alpha} \sigma'}{\rho \parallel \sigma \longrightarrow \rho' \parallel \sigma'}$$

We write \Longrightarrow for the reflexive, transitive closure of \longrightarrow ; we write $\rho \parallel \sigma \dashrightarrow$ if there exist no ρ' and σ' such that $\rho \parallel \sigma \longrightarrow \rho' \parallel \sigma'$. We say that (the client contract) ρ is strongly compliant with (the service contract) σ , notation $\rho \dashv \sigma$, if $\rho \parallel \sigma \Longrightarrow \rho' \parallel \sigma' \dashrightarrow$ implies $\rho' \xrightarrow{e}$.

The first two rules in the definition of \longrightarrow for systems indicate that client and service may evolve autonomously 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 $\rho' \xrightarrow{e}$, which denotes the satisfaction of the client. For instance $a.e + b.e \dashv \bar{a} \oplus \bar{b}$ and $a.e \oplus b.e \dashv \bar{a} + \bar{b}$, but $a.e \oplus b.e \not\vdash \bar{a} \oplus \bar{b}$ because of the computation $a.e \oplus b.e \parallel \bar{a} \oplus \bar{b} \Longrightarrow a.e \parallel \bar{b} \dashrightarrow$.

Observe that compliance is an asymmetric relation as it only concerns the client’s satisfaction. Also, compliance is preserved by system reduction. Namely, if $\rho \dashv \sigma$ and $\rho \parallel \sigma \Longrightarrow \rho' \parallel \sigma'$, then $\rho' \dashv \sigma'$.

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 4 (strong subcontract relation). Let $\llbracket \sigma \rrbracket^s \stackrel{\text{def}}{=} \{\rho \mid \rho \dashv \sigma\}$. We say that σ is a subcontract of τ , notation $\sigma \sqsubseteq \tau$, if $\llbracket \sigma \rrbracket^s \subseteq \llbracket \tau \rrbracket^s$. We write \simeq for the equivalence relation induced by \sqsubseteq , namely $\simeq = \sqsubseteq \cap \supseteq$.

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 $a.(\bar{b} + \bar{d}) \not\sqsubseteq a.\bar{b} + a.\bar{d}$ since $\bar{a}.b.e \dashv a.(\bar{b} + \bar{d})$ but $\bar{a}.b.e \not\vdash a.\bar{b} + a.\bar{d}$ because of the computation $\bar{a}.b.e \parallel a.\bar{b} + a.\bar{d} \longrightarrow b.e \parallel \bar{d} \dashrightarrow$. In the last example a client of $a.(\bar{b} + \bar{d})$ can decide whether to receive \bar{b} or \bar{d} after sending \bar{a} , whereas in $a.\bar{b} + a.\bar{d}$ only one of these actions is available, according to the branch taken by the service. In fact it is possible to prove that $a.\bar{b} + a.\bar{d} \simeq a.(\bar{b} \oplus \bar{d})$.

2.2 Alternative characterization

The set-theoretic definition of the subcontract relation above embeds the notion of safe-substitutability by its own definition, but gives little insight on the properties of \sqsubseteq and is also hard to use in proofs. For these reasons it is desirable to define an alternative characterization of \sqsubseteq , which is also propedeutic to the alternative characterizations of the weak subcontract relations in §3 and in §4. 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 5 (contract continuation). Let $\sigma \xrightarrow{\alpha}$. The continuation of σ with respect to α , notation $\sigma(\alpha)$, is defined as $\sigma(\alpha) \stackrel{\text{def}}{=} \bigoplus_{\sigma \xrightarrow{\alpha} \sigma'} \sigma'$. 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, $a.\bar{b} + a.\bar{d} \xrightarrow{a} \bar{b}$ (the process knows which branch has been taken after an action a) but $(a.\bar{b} + a.\bar{d})(a) = \bar{b} \oplus \bar{d}$ (the party interacting with $a.\bar{b} + a.\bar{d}$ does not know which branch has been taken after seeing an a action, hence it considers both).

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

Definition 6 (ready set). We say that σ has ready set R , written $\sigma \Downarrow R$, if $\sigma \Longrightarrow \sigma' \dashv\dashv$ and $R = \text{init}(\sigma')$.

Intuitively, $\sigma \Downarrow R$ 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 7 (coinductive strong subcontract). We say that \mathcal{S} is a coinductive strong subcontract relation if $(\sigma, \tau) \in \mathcal{S}$ implies

1. $\tau \Downarrow S$ implies $\sigma \Downarrow R$ and $R \subseteq S$ for some R , and
2. $\tau \xrightarrow{\alpha}$ implies $\sigma \xrightarrow{\alpha}$ and $(\sigma(\alpha), \tau(\alpha)) \in \mathcal{S}$.

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.

The next result proves that Definition 7 is indeed a sound and complete characterization of \sqsubseteq .

Theorem 1. \sqsubseteq is the largest coinductive subcontract relation.

2.3 Properties of the strong subcontract relation

The alternative characterization of \sqsubseteq allows us to prove some interesting properties of the strong subcontract relation. First of all, we have now formal evidence that $\sigma \oplus \tau \sqsubseteq \sigma$ holds for every σ and τ . Indeed, the ready sets of σ are a subset of the ready sets of $\sigma \oplus \tau$, hence both conditions in Definition 7 are trivially satisfied. However, \sqsubseteq turns out to be rather restrictive, at least if compared to the hypothetical subcontract relation \preceq we have informally used in the introduction. In particular, we have $a \not\sqsubseteq a + b$, because $a + b$ can offer a b action that is not offered by a , thus violating condition (2). For example, the client $\rho \stackrel{\text{def}}{=} \bar{a}.e + \bar{b}$ is such that $\rho \dashv a$, but $\rho \not\vdash a + b$. Obviously, permutation of actions is also unsupported by \sqsubseteq . Nonetheless, \sqsubseteq enjoys other fundamental properties, and it is at the core of the weak subcontract relations we will define in later sections.

Proposition 1. *The following properties hold:*

1. \sqsubseteq coincides with the must preorder [16, 15, 21] for strongly convergent processes;
2. \sqsubseteq is a precongruence with respect to all the operators of the contract language.

Property (1) connects \sqsubseteq with the well-known *must* testing preorder. This result is not entirely obvious because the notion of “satisfied client” we use for comparing services differs from that of “passing a test” used for comparing processes in the standard testing framework (see [24] for more details).

Property (2) states that \sqsubseteq is well behaved and that it can be used for modular refinement. Namely, we can refine parts of a contract separately, with the guarantee that the resulting contract is a refinement of the original one. The weak variants of the subcontract relation that we will define in the following sections do not enjoy this property in general, but not without reason as we will see.

Further insight on \sqsubseteq can be gained by giving a sound and complete axiomatization of the subcontract relation. This consists of a finite set of *laws* that express the fundamental properties of \preceq and that can be used for proving every relation $\sigma \sqsubseteq \tau$.

Table 1. Axiomatization for \sqsubseteq .

(E1) $\sigma + \sigma = \sigma$	(D1) $\sigma + (\sigma' \oplus \sigma'') = (\sigma + \sigma') \oplus (\sigma + \sigma'')$
(E2) $\sigma + \tau = \tau + \sigma$	(D2) $\sigma \oplus (\sigma' + \sigma'') = (\sigma \oplus \sigma') + (\sigma \oplus \sigma'')$
(E3) $\sigma + (\sigma' + \sigma'') = (\sigma + \sigma') + \sigma''$	(D3) $\alpha.\sigma + \alpha.\tau = \alpha.(\sigma \oplus \tau)$
(E4) $\sigma + 0 = \sigma$	(D4) $\alpha.\sigma \oplus \alpha.\tau = \alpha.(\sigma \oplus \tau)$
(I1) $\sigma \oplus \sigma = \sigma$	(RED) $\sigma \oplus \tau \leq \sigma$
(I2) $\sigma \oplus \tau = \tau \oplus \sigma$	
(I3) $\sigma \oplus (\sigma' \oplus \sigma'') = (\sigma \oplus \sigma') \oplus \sigma''$	

Table 1 defines an axiomatization for the relation \leq , which coincides with \sqsubseteq as we will see shortly. In the table we use a single axiom $\sigma = \tau$ in place of two axioms $\sigma \leq \tau$ and $\tau \leq \sigma$. Rules (E1–E4) state that the external choice is an idempotent, commutative,

and associative operator with neutral element 0. Rules (I1–I3) state that internal choice is idempotent, commutative, and associative. Rules (D1–D2) state that the two choices distribute over each other, while rules (D3–D4) state the distributivity laws of prefix over choices. Rules (D3) and (D4) together state a particularly important fact: an external choice $\alpha.\sigma + \alpha.\tau$ where both branches are guarded by the *same* action is actually an internal choice $\alpha.\sigma \oplus \alpha.\tau$ in disguise. This is one of the reasons why $\sigma \leq \sigma + \tau$ does *not* hold in general, the other being that τ can introduce interferences as we have seen in an earlier example. Rule (RED) is perhaps the most important one in that it characterizes the essence of \sqsubseteq as a relation that reduces nondeterminism.

The axiomatization in Table 2 is sound and complete with respect to \sqsubseteq .

Theorem 2. $\sqsubseteq = \leq$.

The proof of this result is omitted since in §3 we are going to prove soundness and completeness of the deduction system for a weaker subcontract relation that includes \sqsubseteq at its core.

3 Towards a weaker subcontract relation

3.1 Synchronous orchestrators

We have seen that the strong subcontract relation (Definition 4) is quite restrictive in the contracts that it deems comparable and the reason lies, not surprisingly, in the compliance relation (Definition 3) from which it is defined. Thus, we need to re-consider the compliance relation and in particular we have to investigate in more details the *reasons why* some clients are not satisfied by some services. Let us start by considering the interaction

$$\bar{a}.e \oplus \bar{b}.e \parallel a \longrightarrow \bar{b}.e \parallel a \quad (1)$$

which tells us that the client $\bar{a}.e \oplus \bar{b}.e$ is not compliant with the service a . In this example we could say that the client is the one to blame, since it may autonomously decide to send a message b that the service is not willing to receive. We could also say that the service is the one to blame, since it is not flexible enough to receive every message that the client is willing to send. In a dual manner, we can consider the interaction

$$\bar{a}.e \parallel a \oplus b \longrightarrow \bar{a}.e \parallel a \quad (2)$$

which tells us that the client $\bar{a}.e$ is not compliant with the service $a \oplus b$. In this case too we can put the blame on either the client or the service, but the point we want to make here is that in both examples the missed compliance is a consequence of some *internal* behavior of either (or both) parties, and that there is no way to mend the situation, since we assume that clients and services are black boxes whose internal behavior cannot be constrained from the outside.

Consider now the interaction

$$\bar{a}.e + \bar{b}.c.e \parallel a + b.\bar{d} \longrightarrow c.e \parallel \bar{d}$$

which proves that $\bar{a}.e + \bar{b}.c.e$ is not compliant with $a + b.\bar{d}$. If we were to describe contracts using English words, it is as if the client says: “here are a message a and a message b , please pick one. If you choose the message b , send me back a message c ”. On the other side, the service says: “I’m willing to receive a message a or a message b , please choose what you want. If you choose the message b , I will send you back a message d ”. We realize that this example of failed compliance differs significantly from the previous ones, because the failure is due to a choice that is *external* from both the client and the service. None of them mandates a particular synchronization, on a or on b , to occur, but the strong compliance takes into account *every possible synchronization*, and in the interaction above the bad one has been chosen. Had client and service synchronized on a , which was a perfectly feasible option for both the client and the service, the client would have been satisfied.

The last example provides us with the necessary enlightenment for relaxing the definition of compliance: if there is a third party that controls the interaction between the client and the service, in the sense that it may prevent certain synchronizations from happening, then $\bar{a}.e + \bar{b}.c.e$ *can be made* compliant with $a + b.\bar{d}$. In the Web service domain, this centralized control of several interacting processes is called orchestration, and the controller is itself a process called orchestrator. Here we consider a simpler scenario in which we have only two interacting processes, the client and the service, and where the orchestrator can only prevent some synchronizations from occurring. We will relax this latter assumption in §4. For the time being, we will use the term *synchronous orchestrator* to indicate an orchestrator that is only capable of enabling or disabling some synchronizations.

It is worth to remark that in no way an orchestrator can affect the internal choices of a process. For example, there is no orchestrator that can make the clients of the interactions (1) and (2) compliant with the corresponding services.

Having realized that we might need an orchestrator to enforce the compliance relation, we now proceed into formalizing the language of synchronous orchestrators. It turns out that the orchestrators required for fixing the kind of problems depicted above are made of orchestration actions having one of the following two forms:

$$\langle a, \bar{a} \rangle \quad \text{or} \quad \langle \bar{a}, a \rangle$$

An orchestration action of the form $\langle a, \bar{a} \rangle$ permits a synchronization between client and service provided that the client is ready to send a message on a and the service is ready to receive it. An orchestration action of the form $\langle \bar{a}, a \rangle$ permits the dual synchronization, in which the client is ready to receive a message on a and the service is ready to send it. It is as if the orchestrator is a process with two distinct interfaces, one that connects with the client, the other that connects with the service. Orchestration actions $\langle \alpha, \bar{\alpha} \rangle$ show which actions the orchestrator simultaneously offers on both interfaces. These orchestration actions are called *synchronous* because they permit client and service to synchronize when *both* parties are ready to perform an action and the corresponding co-action. We will extend filters to asynchronous actions in §4.

Definition 8 (synchronous orchestrators). Synchronous orchestrators, ranged over by f, g, \dots , are described by the following grammar:

$$\begin{array}{ll}
 f ::= & \mathbf{orchestrator} \\
 0 & (\text{null}) \\
 | \mu.f & (\text{action prefix}) \\
 | f \vee g & (\text{disjunction})
 \end{array}
 \qquad
 \begin{array}{ll}
 \mu ::= & \mathbf{action} \\
 \langle a, \bar{a} \rangle & (\text{input/output}) \\
 | \langle \bar{a}, a \rangle & (\text{output/input})
 \end{array}$$

The null orchestrator 0 is the one that does not permit any synchronization between client and service. The orchestrator $\mu.f$ allows the orchestration action μ . If this action is performed, it continues as the orchestrator f . The orchestrator $f \vee g$ is the *disjunction* of the orchestrators f and g : it permits all orchestration actions that are permitted by either f or by g . As for contracts we will omit trailing 0 's in orchestrators.

In our development orchestrators do not exhibit internal nondeterminism. This calls for an operational semantics merely expressing which orchestration actions are available.

Definition 9 (operational semantics of orchestrators). The operational semantics of orchestrators is inductively defined by the rules below

$$\mu.f \xrightarrow{\mu} f \qquad \frac{f \xrightarrow{\mu} f'}{f \vee g \xrightarrow{\mu} f'} \qquad \frac{g \xrightarrow{\mu} g'}{f \vee g \xrightarrow{\mu} g'}$$

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

$$\llbracket f \rrbracket \stackrel{\text{def}}{=} \{ \mu_1 \dots \mu_n \mid \exists g : f \xrightarrow{\mu_1} \dots \xrightarrow{\mu_n} g \}$$

Observe that $\llbracket f \rrbracket$ is a non-empty, prefix-closed, set of strings over the alphabet of orchestration actions and that \vee corresponds to a union operator on the traces of orchestrators, namely $\llbracket f \vee g \rrbracket = \llbracket f \rrbracket \cup \llbracket g \rrbracket$. We write $f \xrightarrow{\mu} g$ if $\llbracket g \rrbracket = \{ \mu_1 \dots \mu_n \mid \mu \mu_1 \dots \mu_n \in \llbracket f \rrbracket \}$, namely g is the residual of f after the action μ . Note that $f \xrightarrow{\mu} f'$ and $f \xrightarrow{\mu} f''$ implies $\llbracket f' \rrbracket = \llbracket f'' \rrbracket$. We write $f \xrightarrow{\mu_1 \dots \mu_n}$ if $\mu_1 \dots \mu_n \in \llbracket f \rrbracket$; we write $f \xrightarrow{\mu}$ if $\mu \notin \llbracket f \rrbracket$; let $\text{init}(f) \stackrel{\text{def}}{=} \{ \mu \mid f \xrightarrow{\mu} \}$.

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 10 (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:

$$\frac{\rho \longrightarrow \rho'}{\rho \parallel_f \sigma \longrightarrow \rho' \parallel_f \sigma} \qquad \frac{\sigma \longrightarrow \sigma'}{\rho \parallel_f \sigma \longrightarrow \rho \parallel_f \sigma'} \qquad \frac{\rho \xrightarrow{\bar{\alpha}} \rho' \quad f \xrightarrow{\langle \alpha, \bar{\alpha} \rangle} f' \quad \sigma \xrightarrow{\alpha} \sigma'}{\rho \parallel_f \sigma \longrightarrow \rho' \parallel_{f'} \sigma'}$$

We write \Longrightarrow for the reflexive, transitive closure of \longrightarrow ; we write $\rho \parallel_f \sigma \dashv\vdash$ if there exist no ρ' , f' , and σ' such that $\rho \parallel_f \sigma \longrightarrow \rho' \parallel_{f'} \sigma'$. We write $f : \rho \dashv\vdash \sigma$ if $\rho \parallel_f \sigma \Longrightarrow \rho' \parallel_{f'} \sigma' \dashv\vdash$ implies $\rho' \xrightarrow{e}$. We say that ρ is weakly compliant with σ , notation $\rho \dashv\vdash \sigma$, if there exists an orchestrator f such that $f : \rho \dashv\vdash \sigma$.

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 $\langle \alpha, \bar{\alpha} \rangle$ “connects” the action $\bar{\alpha}$ performed by the client with the action α performed by the service). For example, we have $\langle a, \bar{a} \rangle : \bar{a}.e + \bar{b}.c.e \dashv\vdash a + b.\bar{d}$ because the orchestrator $\langle a, \bar{a} \rangle$ disallows the synchronization on b at the first step while permitting the synchronization on a . At the same time $\bar{a}.e \oplus \bar{b}.e \not\vdash a$ because no orchestrator can prevent the client from autonomously evolving to $\bar{b}.e$.

Weak compliance induces the weak subcontract relation as follows:

Definition 11 (weak subcontract). Let $\llbracket \sigma \rrbracket^w \stackrel{\text{def}}{=} \{ \rho \mid \rho \dashv\vdash \sigma \}$. We say that σ is a weak subcontract of τ , notation $\sigma \preceq \tau$, if $\llbracket \sigma \rrbracket^s \subseteq \llbracket \tau \rrbracket^w$.

According to this definition, $\sigma \preceq \tau$ holds whenever every client satisfied by σ (that is $\rho \dashv\vdash \sigma$) can also be satisfied by τ (that is $\rho \dashv\vdash \tau$) by means of some orchestrator f . So, it is safe to replace σ with τ , provided that f mediates the interaction between the client and τ .

Whether or not \preceq is the subcontract relation we are looking for is hard to tell from Definition 11. In part this is because it is very reasonable to expect that the 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 subsection 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. In order to define relevant orchestrators, we need three auxiliary notions. The first one is the *identity orchestrator* $I(\sigma)$ for a contract σ , which is the orchestrator that enables all the traces of actions in σ :

$$I(\sigma) \stackrel{\text{def}}{=} \bigvee_{\sigma \xrightarrow{\alpha}} \langle \alpha, \bar{\alpha} \rangle . I(\sigma(\alpha)) \quad (3)$$

The second notion we need is a derived operator \wedge over orchestrators that is dual of \vee . We define

$$f \wedge g \stackrel{\text{def}}{=} \bigvee_{f \xrightarrow{\mu}, g \xrightarrow{\mu}} \mu . (f' \wedge g')$$

It is trivial to see that $\llbracket f \wedge g \rrbracket = \llbracket f \rrbracket \cap \llbracket g \rrbracket$. The third and last notion is an ordering that allows us to compare orchestrators. We define

$$f \leq g \stackrel{\text{def}}{\iff} \llbracket f \rrbracket \subseteq \llbracket g \rrbracket$$

namely $f \leq g$ holds if every sequence of orchestration actions offered by f is also offered by g . Using this ordering, we see that $f \vee g$ and $f \wedge g$ respectively correspond to the least upper bound and to the greatest lower bound of f and g .

Definition 12 (relevant orchestrator). *Let $\sigma \preceq \tau$ and f be an orchestrator. We say that f is relevant for $\sigma \preceq \tau$ if $f \leq I(\sigma) \wedge I(\tau)$.*

An orchestrator that is relevant for $\sigma \preceq \tau$ never offers orchestration actions that do not correspond to actions offered by σ or that would never be enabled by τ . It is easy to see that, given an orchestrator f such that $f : \rho \dashv\vdash \tau$, then $f \wedge I(\sigma) \wedge I(\tau) : \rho \dashv\vdash \tau$ where $f \wedge I(\sigma) \wedge I(\tau)$ is relevant for $\sigma \preceq \tau$.

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 13 (universal orchestrator). *We say that f is a universal orchestrator proving $\sigma \preceq \tau$, notation $f : \sigma \preceq \tau$, if $\rho \dashv\vdash \sigma$ implies $f : \rho \dashv\vdash \tau$ 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. The next result assures us that if $\sigma \preceq \tau$, then a universal orchestrator proving the relation always exists.

Proposition 2. *$\sigma \preceq \tau$ if and only if $f : \sigma \preceq \tau$ for some orchestrator f .*

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, if we think of contracts as of (behavioral) types, the orchestrator f is an *explicit coercion* [8, 11, 29] that maps 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 14 (orchestrator application). *The application of the orchestrator f to the contract σ , notation $f(\sigma)$, is defined as*

$$f(\sigma) \stackrel{\text{def}}{=} \bigoplus_{\sigma \Downarrow \mathbb{R}} \sum_{f(\alpha, \bar{\alpha}) \xrightarrow{f'} \alpha, \alpha \in \mathbb{R}} \alpha.f'(\sigma(\alpha))$$

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. *$f : \rho \dashv\vdash \sigma$ if and only if $\rho \dashv\vdash f(\sigma)$.*

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

Corollary 1. *$f : \sigma \preceq \tau$ if and only if $\sigma \sqsubseteq f(\tau)$.*

Proof. By Theorem 3 $f : \sigma \preceq \tau$ if and only if $\rho \dashv \sigma$ implies $f : \rho \dashv \tau$ if and only if $\rho \dashv \sigma$ implies $\rho \dashv f(\sigma)$ if and only if $\sigma \sqsubseteq f(\tau)$. \square

Corollary 1 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 and that \preceq permits width and depth extensions:

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

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. *The following properties hold:*

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

Composition of synchronous orchestrators. Transitivity of the weak subcontract relation is not granted by Definition 11, 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 1 we know that $\sigma \sqsubseteq f(\tau)$ and $\tau \sqsubseteq g(\sigma')$. Furthermore, by Proposition 3(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. The next result, whose proof is left as an easy exercise, confirms that this is indeed the case and that \wedge corresponds to the functional composition operator between synchronous orchestrators.

Proposition 4. $f(g(\sigma)) \simeq (f \wedge g)(\sigma)$.

3.3 Alternative characterization of the weak subcontract relation

We now proceed to define an alternative, coinductive characterization of \preceq , as we have done for \sqsubseteq . Observe that, when an orchestrator mediates the interaction between a client and a service, it proposes at each interaction step a set of orchestration actions A whose effect is to filter out some actions offered by either the client or by the service. Since a synchronization occurs only if both client and service are willing to interact on corresponding co-actions, it is sufficient to consider the effect of the orchestrator on only one of the two interacting parties, which we take to be the service.

If S is a service ready set, then $A \circ S$ denotes the ready set filtered by the orchestrator, as perceived by the client:

$$A \circ S \stackrel{\text{def}}{=} \{\alpha \in S \mid \langle \alpha, \bar{\alpha} \rangle \in A\}$$

Namely, the client sees an action α only if that action is provided by the service ($\alpha \in S$) and the orchestrator does not hide it ($\langle \alpha, \bar{\alpha} \rangle \in A$).

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

Definition 15 (coinductive weak subcontract). *We say that \mathcal{W} is a coinductive weak subcontract relation if $(\sigma, \tau) \in \mathcal{W}$ implies that there exists a set of orchestration actions A such that*

1. $\tau \Downarrow S$ implies $\sigma \Downarrow R$ and $R \subseteq A \circ S$ for some R , and
2. $\tau \xrightarrow{\alpha}$ and $\langle \alpha, \bar{\alpha} \rangle \in A$ implies $\sigma \xrightarrow{\alpha}$ and $(\sigma(\alpha), \tau(\alpha)) \in \mathcal{W}$.

Condition (1) requires that τ must look like a more deterministic version of σ when filtered by the orchestrator (the ready set $A \circ S$ of the orchestrated service has a corresponding one of σ that offers fewer actions). Condition (2) poses the usual requirement that the continuations must be in the subcontract relation, but only for those actions that are permitted by the orchestrator. Observe that the set A of orchestration actions must be defined independently of the internal state of the larger contract: the implicit universal quantifier over the ready sets S of τ in condition (1) falls within the scope of the existential quantifier that binds A . This enforces the fact that the orchestrator treats τ as a black box, and that it is not able to sense its internal state.

The two definitions of weak subcontract are equivalent:

Theorem 4. \preceq is the largest coinductive weak subcontract relation.

3.4 Deduction system for the weak subcontract relation

Unlike the strong subcontract relation, \preceq is *not* a precongruence with respect to the external choice. For example, we have

$$0 : 0 \preceq a.c \quad \text{and} \quad \langle a, \bar{a} \rangle . \langle b, \bar{b} \rangle : a.b \preceq a.b$$

but $a.b + 0 \not\preceq a.b + a.c$ since $a.b + a.c \simeq a.(b \oplus c)$. However, there is actually a good reason why \preceq is not a precongruence in general. The relation $0 \preceq a.c$ holds because the 0 orchestrator turns a service with contract $a.c$ into a service with contract 0, and the identity $a.b \preceq a.b$ holds because of the orchestrator $I(a.b)$. When we combine $a.b$ and 0 into $a.b + 0$ we are creating a new service that externally offers either the $a.b$ behavior or the null one. When this service interacts with a client, it must do so by means of an orchestrator, but which one? The problem is that we need two different orchestrators for relating the two branches of the external choice, and there is no single orchestrator that works equally well for both branches, in particular $0 \wedge I(a.b) : a.b \not\preceq a.b$ and $0 \vee I(a.b) : 0 \not\preceq a.c$.

As a direct consequence of the lack of the pre-congruence property for \preceq , we have that the axiomatization of \preceq is more challenging than that of \sqsubseteq . Since \preceq is not a pre-congruence, we cannot “substitute equals for equals” when applying the rewriting rules. This means that we have to look for the conditions under which substitution in a context is safe, and to explicitly provide pre-congruence deduction rules that enforce these conditions. Luckily, all the additional information we need is stored in the orchestrator that proves a relation $\sigma \preceq \tau$. This allows us to design a deduction system for judgments of the form

$$f : \sigma \leq \tau$$

where f is the orchestrator that “proves” the relation $\sigma \leq \tau$.

Table 2. Deduction system for the weak subcontract relation.

(RED) $I(\sigma) : \sigma \oplus \tau \leq \sigma$	(PREFIX) $\frac{f : \sigma \leq \tau}{\langle \alpha, \bar{\alpha} \rangle . f : \alpha . \sigma \leq \alpha . \tau}$
(DEPTH) $0 : 0 \leq \sigma$	(INT) $\frac{f : \sigma \leq \sigma' \quad f : \tau \leq \tau'}{f : \sigma \oplus \tau \leq \sigma' \oplus \tau'}$
(WEAK) $\frac{f : \sigma \leq \tau \quad g \wedge I(\tau) \leq f}{f \vee g : \sigma \leq \tau}$	(EXT) $\frac{f : \sigma \leq \sigma' \quad f : \tau \leq \tau'}{f : \sigma + \tau \leq \sigma' + \tau'}$
(TRANS) $\frac{f : \sigma \leq \sigma' \quad g : \sigma' \leq \sigma''}{f \wedge g : \sigma \leq \sigma''}$	

The deduction system for \preceq is defined by the equalities in Table 1 and the axioms and rules in Table 2. Every equality $\sigma = \tau$ inherited from Table 1 is meant to be a shorthand for two axioms $I(\tau) : \sigma \leq \tau$ and $I(\sigma) : \tau \leq \sigma$. Let us comment the remaining deduction rules. Rule (RED) is the same as in Table 1, except that the relation is now witnessed by the orchestrator $I(\sigma)$. In fact, this relation holds without the need of filtering any action, hence the orchestrator $I(\sigma \oplus \tau)$ would work as well. However, the rule as is stated has the advantage that $I(\sigma)$ is relevant for $\sigma \oplus \tau \leq \sigma$, and the judgment with orchestrator $I(\sigma \oplus \tau)$ is derivable by means of (WEAK), which we are to describe shortly. Rule (DEPTH) expresses the fact that the weak subcontract relation always supports depth extensions of contracts, by means of the null orchestrator which prevents any interference caused by the larger σ contract. Thus, 0 is the least element for \preceq . Rule (WEAK) is a sort of weakening rule that allows less restrictive orchestrators to be used without compromising safety. The rule says that if f proves $\sigma \leq \tau$, then $f \vee g$ also proves the same relation provided that $f \vee g$ does not enable any action from τ that must be kept hidden in order for $\sigma \leq \tau$ to hold. This is guaranteed if the projection of g on the traces of τ ($g \wedge I(\tau)$) is a subset of the traces of actions that are already enabled by f . Rule (TRANS) is the standard transitivity rule, where the resulting orchestrator is the composition of the two orchestrators in the premises of the rule (see Proposition 4). On the right hand side of Table 2 we have three rules of (restricted) pre-congruence. Rule (PREFIX) shows that \leq is a pre-congruence with respect to prefixes. Rules (INT)

and (EXT) state restricted precongruence of \leq with respect to the two choices. In both cases the requirement is that both branches of a choice must be orchestrated in the same way.

The rest of this subsection is devoted to proving that the deduction system is sound and complete. Observe that soundness and completeness of the deduction system now have a stronger meaning since they must take into account the fact that judgments $f : \sigma \leq \tau$ include an orchestrator f . Hence, “soundness” means not only that if $f : \sigma \leq \tau$, then $\sigma \preceq \tau$, but also that f is an orchestrator that proves the relation. At the same time, “completeness” means that if f is an orchestrator proving $\sigma \preceq \tau$, then f can be obtained by means of the rules in the deduction system.

Theorem 5 (soundness). *If $f : \sigma \leq \tau$, then $f : \sigma \preceq \tau$.*

As usual soundness of the deduction system is relatively straightforward, whereas completeness is much harder to prove. The completeness proof relies on the ability to rewrite a contract, by means of the axioms in Table 1, in a so-called *normal form*, which is a syntactic, canonical representation of the contract’s semantics. Once two contracts are in normal-form, checking that they are related is a much simpler matter. Much of the development that follows is an adaptation of the completeness proof of the must testing preorder found in [21].

The normal form of a contract arises by observing that the set of ready sets of a contract can be saturated without changing the semantics of a contract, according to the strong subcontract relation. More precisely, we close ready sets by union and by convex closure. For example, we have $a \oplus b \simeq a \oplus b \oplus (a + b)$, where we have added the ready set $\{a, b\}$ as the union of the ready sets $\{a\}$ and $\{b\}$ of the contract on the l.h.s. of \simeq . As another example, we have $a \oplus (a + b + c) \simeq a \oplus (a + b) \oplus (a + c) \oplus (a + b + c)$, where we have added the ready sets $\{a, b\}$ and $\{a, c\}$ which include $\{a\}$ and are included in $\{a, b, c\}$. The usefulness of saturation comes from the fact that once two contracts have been saturated, condition (1) of Definition (7) reduces to verifying that every ready set of the larger contract is also a ready set of the smaller contract. This makes the application of rule (RED) in the completeness proof (almost) straightforward.

Definition 16 (contract normal form [21]). *The saturated set of ready sets of a contract σ , notation $\mathcal{R}(\sigma)$, is defined as $\mathcal{R}(\sigma) \stackrel{\text{def}}{=} \{R \subseteq \text{init}(\sigma) \mid \exists S : \sigma \Downarrow S \wedge S \subseteq R\}$. We say that a contract σ is in normal form if $\sigma \equiv \bigoplus_{R \in \mathcal{R}(\sigma)} \sum_{\alpha \in R} \alpha. \sigma_\alpha$ where every continuation σ_α is itself in normal form and \equiv denotes syntactic equality up to associativity and commutativity of the choice operators.*

Observe that if a contract σ is in normal form, the residual of the contract after a visible action is unique, namely $\sigma \xRightarrow{\alpha} \sigma'$ and $\sigma \xRightarrow{\alpha} \sigma''$ implies $\sigma' \equiv \sigma''$.

Following [21], we derive a bunch of handy axioms and rules (Table 3). Axioms (S1) and (S2) implement the saturation of ready set as described above. Axiom (CO) is used to combine the residual of a contract with respect to some given action α . Rules (E-PREFIX), (E-INT), and (E-EXT) respectively specialize rules (PREFIX), (INT), and (EXT) so as to make it possible the substitution of equals for equals in arbitrary contexts. Finally, rule (WIDTH) embeds width extensions of contracts, provided that the additional capabilities (in τ) do not interfere with the old ones (in σ). This is implied by the fact

that the identity orchestrator for σ shares no common trace with the identity orchestrator for τ , except for ε ($I(\sigma) \wedge I(\tau) \leq 0$).

Table 3. Derived rules.

(S1)	$\sigma \oplus \tau = \sigma \oplus \tau \oplus (\sigma + \tau)$	(E-PREFIX)	$\frac{\sigma = \sigma'}{\alpha.\sigma = \alpha.\sigma'}$
(S2)	$\sigma \oplus (\sigma + \tau + \rho) = \sigma \oplus (\sigma + \tau) \oplus (\sigma + \tau + \rho)$		
(CO)	$(\alpha.\sigma' + \tau') \oplus (\alpha.\sigma'' + \tau'') =$ $(\alpha.(\sigma' \oplus \sigma'') + \tau') \oplus (\alpha.(\sigma' \oplus \sigma'') + \tau'')$	(E-EXT)	$\frac{\sigma = \sigma' \quad \tau = \tau'}{\sigma + \tau = \sigma' + \tau'}$
(WIDTH)	$\frac{I(\sigma) \wedge I(\tau) \leq 0}{I(\sigma) : \sigma \leq \sigma + \tau}$	(E-INT)	$\frac{\sigma = \sigma' \quad \tau = \tau'}{\sigma \oplus \tau = \sigma' \oplus \tau'}$

Lemma 1. *The axioms and rules in Table 3 can be derived from those in Tables 1 and 2.*

The last auxiliary result is the one assuring us that every contract can be rewritten into a \simeq -equivalent one that is in normal form.

Lemma 2 (normal form). *For every contract σ there exists σ' in normal form such that $\sigma = \sigma'$.*

The deduction system for \preceq defined by Tables 1 and 2 is complete for \preceq and the sets of filters that prove it.

Theorem 6 (completeness). *If $f : \sigma \preceq \tau$, then $f : \sigma \leq \tau$.*

3.5 Interpretations of synchronous orchestrators

At the beginning of this section we have introduced orchestrators as centralized control points mediating the interaction between clients and services. Orchestrators are limited in that they can only affect the way client and service try to interact with each other, but they cannot affect in any way the internal moves of clients and services. This is clearly reflected in the transition relation for orchestrated systems (Definition 10) and more technically in the coinductive characterization of the weak subcontract relation (Definition 15).

Corollary 1 provides us with another interpretation of orchestrators: they are morphisms that transform services into services with a (slightly) different contract. In this interpretation an orchestrator is like an explicit behavioral coercion that is applied to (wrapped around) a process so as to change its contract. Along the section we have made the implicit assumption that the coercion is applied to the service, because this view has allowed us to develop a theory of synchronous orchestrators that is oblivious to the specific client that we are considering. In practice, it is perhaps more reasonable to expect that the morphism is applied to the client, which is the one that must be satisfied.

When interpreting orchestrators as mediators or as morphisms, the query to a registry of Web service should return not only the services that satisfy the client with contract ρ , but also the orchestrator that ensures the successful interaction. Namely, the query looks like this:

$$\mathcal{Q}(\rho) = \{(f, \sigma) \mid f : \rho^\perp \preceq \sigma\}$$

There is a further interpretation of orchestrators as abstract specifications of those clients that can interact successfully with a given service. Consider for example a relation $f : \sigma \preceq \tau$ that is proved by an orchestrator f that never hides any action from τ . Any further action that the orchestrator may allow is practically useless, hence by replacing σ with τ the orchestrator is never actually preventing any synchronization between the client and the service with contract τ . This is formalized by the following proposition, which gives a sufficient (and also necessary, for that matters) condition when the weak subcontract relation reduces to the strong one:

Proposition 5. *If $f : \sigma \preceq \tau$ and $I(\tau) \leq f$, then $\sigma \sqsubseteq \tau$.*

A finer control can be implemented if the contract of the client performing the query is known. Suppose for example that the client contract is ρ and that the Web service repository contains a registered service whose contract is σ , where $f : \rho^\perp \preceq \sigma$. In principle, the client is satisfied by the service provided that the orchestrator f mediates the interaction between the two parties. However, suppose in addition that the orchestrator never hides any action that the client is offering to the service. This can be expressed as $I(\rho^\perp) \leq f$. Then, the particular client with contract ρ is also *strongly compliant* with the service with contract σ , and no orchestration is actually necessary. This is formalized by the following proposition:

Proposition 6. *If $f : \rho^\perp \preceq \sigma$ and $I(\rho^\perp) \leq f$, then $\rho \dashv \sigma$.*

For example, consider a client with contract $\rho \stackrel{\text{def}}{=} \bar{a}.b.e$ and suppose that the registry contains a Web service with contract $a.\bar{b} + c.\bar{d}$. Although we have not seen how to compute the dual of the client's contract yet, in this case it is intuitively obtained by simply swapping inputs and outputs. Namely, $\rho^\perp = a.\bar{b}$ (the e action disappears as it is only used for denoting client's satisfaction). We observe that $I(\rho^\perp) = \langle a, \bar{a} \rangle. \langle \bar{b}, b \rangle$, and that $I(\rho^\perp) : \rho^\perp \preceq a.\bar{b} + c.\bar{d}$, hence by Proposition 6 we can conclude that $\rho \dashv a.\bar{b} + c.\bar{d}$, without the intervention of any orchestrator at all. Observe however that $\rho^\perp \not\sqsubseteq a.\bar{b} + c.\bar{d}$, so the service with contract $a.\bar{b} + c.\bar{d}$ would not be retrieved if the strong subcontract relation were used instead of the weak one.

4 Asynchronous orchestrators

In the previous section we have resorted to the use of an orchestrator that guarantees the successful termination of the client. It is then natural to investigate whether, by augmenting the capabilities of the orchestrator, it is possible to further enlarge the spectrum of services that can satisfy a given client. We observe that both the strong and the weak compliance relations (Definitions 3 and 10) are based on interactions where at each synchronization progress is always guaranteed for *both* client and service. In this section

we relax this requirement and assume that the orchestrator that mediates the interaction of a client and a service 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, nor it can pretend to send a message to a party if that message has not been previously received from the other party.

4.1 Buffered compliance and subcontract relations

We extend orchestration actions so that they are described by the following grammar:

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

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. 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, $\langle \alpha, \bar{\alpha} \rangle$ are *synchronous* orchestration actions because, if executed, they guarantee simultaneous progress to both client and service.

A *directional buffer* is a map $\{\circ, \bullet\} \times \overline{\mathcal{N}} \rightarrow \mathbb{Z}$ associating pairs (r, \bar{a}) with the number of \bar{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 $\mathbb{B}, \mathbb{B}', \dots$ 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 $\text{cod}(\mathbb{B})$ – the codomain of \mathbb{B} – to range over \mathbb{Z} , although every well-formed buffer will always contain a non-negative number of messages. We write $\tilde{\emptyset}$ for the empty buffer, the one having $\{0\}$ as codomain; we write $\mathbb{B}[(r, \bar{a}) \mapsto n]$ for the buffer \mathbb{B}' which is the same as \mathbb{B} except that (r, \bar{a}) is associated with n ; we write $\mathbb{B}\mu$ for the buffer \mathbb{B} updated after the orchestration action μ :

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

We say that \mathbb{B} has rank k , or is a k -buffer, if $\text{cod}(\mathbb{B}) \subseteq [0, k]$; 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 $\tilde{\emptyset} \not\vdash_k \langle \bar{a}, \varepsilon \rangle$ because $-1 \in \text{cod}(\tilde{\emptyset}\langle \bar{a}, \varepsilon \rangle)$. We extend the notion to sets of actions so that $\mathbb{B} \vdash_k A$ 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 remains unchanged, except that now μ ranges over synchronous as well as asynchronous orchestration actions. The operational and

denotation semantics of orchestrators with asynchronous orchestration actions are simple extensions of those with only synchronous orchestration actions and will not be repeated here.

We say that f is a *valid orchestrator of rank k* , or is a *k -orchestrator*, if $f \vdash^{\mu_1 \cdots \mu_n}$ implies that $\tilde{0}\mu_1 \cdots \mu_n$ is a k -buffer. Not every term f denotes a valid orchestrator of finite rank. For instance $\langle a, \varepsilon \rangle . \langle a, \varepsilon \rangle$ is not a valid orchestrator of rank 1 because it accepts two a messages from the client without delivering them to the service; it is, however, a valid orchestrator of rank 2; $\langle \bar{a}, \varepsilon \rangle$ is invalid because it tries to deliver to the client a message that it has not received from the service; symmetrically, $\langle \varepsilon, \bar{a} \rangle$ is invalid because it tries to deliver to the service a message that it has not received from the client; finally, $\langle \varepsilon, a \rangle . \langle \bar{a}, \varepsilon \rangle$ is a valid orchestrator of rank 1 (or greater). In the following we will always work with valid orchestrators of finite rank.

We now proceed to extending weak compliance (Definition 10) to asynchronous orchestrators.

Definition 17 (weak k -compliance relation). A k -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 a k -orchestrator f . Let \longrightarrow be the least relation between orchestrated systems inductively defined as follows:

$$\frac{\rho \longrightarrow \rho'}{\rho \parallel_f \sigma \longrightarrow \rho' \parallel_f \sigma} \quad \frac{\sigma \longrightarrow \sigma'}{\rho \parallel_f \sigma \longrightarrow \rho \parallel_f \sigma'} \quad \frac{\rho \xrightarrow{\bar{\alpha}} \rho' \quad f \xrightarrow{\langle \alpha, \bar{\alpha} \rangle} f' \quad \sigma \xrightarrow{\alpha} \sigma'}{\rho \parallel_f \sigma \longrightarrow \rho' \parallel_{f'} \sigma'}$$

$$\frac{\rho \xrightarrow{\bar{\alpha}} \rho' \quad f \xrightarrow{\langle \alpha, \varepsilon \rangle} f'}{\rho \parallel_f \sigma \longrightarrow \rho' \parallel_{f'} \sigma} \quad \frac{f \xrightarrow{\langle \varepsilon, \bar{\alpha} \rangle} f' \quad \sigma \xrightarrow{\alpha} \sigma'}{\rho \parallel_f \sigma \longrightarrow \rho \parallel_{f'} \sigma'}$$

The transitive closure of \Longrightarrow as well as the usual notation built on top of \longrightarrow is the same as in Definition 10. We say that ρ is weakly k -compliant with σ , notation $\rho \dashv_k \sigma$, if there exists a k -orchestrator f such that $f : \rho \dashv \sigma$.

The first three rules in the definition of \longrightarrow for orchestrated systems are exactly the same as for the weak compliance relation (Definition 10) and deserve no further comment. 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, and that weak compliance and weak 0-compliance do coincide.

As an example of weak k -compliance we have $\bar{a}.\bar{c}.b.e \dashv_k c.a.\bar{b}$, by means of the orchestrator $\langle a, \varepsilon \rangle . \langle c, \varepsilon \rangle . \langle \varepsilon, \bar{c} \rangle . \langle \varepsilon, \bar{a} \rangle . \langle \bar{b}, b \rangle$ which accepts the messages in the order required by the client, but delivers them in the order expected by the service. However, we have $a.\bar{c}.e \not\vdash_k c.\bar{a}$ for every k , since no k -orchestrator is capable of creating the a message that the client is waiting for, before delivering c to the service.

Weak k -compliance induces the weak k -subcontract relation in a similar way as weak compliance induces weak subcontract:

Definition 18 (weak k -subcontract). Let $\llbracket \sigma \rrbracket_k^w \stackrel{\text{def}}{=} \{ \rho \mid \rho \dashv_k \sigma \}$. We say that σ is a weak k -subcontract of τ , notation $\sigma \preceq_k \tau$, if $\llbracket \sigma \rrbracket_k^s \subseteq \llbracket \tau \rrbracket_k^w$.

As for the weak subcontract, this definition says little about the properties of \preceq_k , hence we will go through a similar sequence of preliminary results. Luckily, all the properties enjoyed by \preceq are also valid, in their essence, for \preceq_k . However, the proofs of the results are sometimes more involved because of buffering.

4.2 Basic properties of the weak k -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 19 (relevant k -orchestrator). Let $\sigma \preceq_k \tau$ and f be a k -orchestrator. We say that f is relevant for $\sigma \preceq_k \tau$ if $\sigma \xrightarrow{\varphi_1 \cdots \varphi_n}$ and $f \xrightarrow{\langle \varphi_1, \bar{\varphi}'_1 \rangle \cdots \langle \varphi_n, \bar{\varphi}'_n \rangle \langle \varphi, \bar{\varphi}' \rangle}$ and $\tau \xrightarrow{\varphi'_1 \cdots \varphi'_n}$ imply $\sigma(\varphi_1 \cdots \varphi_n) \xrightarrow{\varphi}$ and $\tau(\varphi'_1 \cdots \varphi'_n) \xrightarrow{\varphi'}$.

A k -orchestrator that is relevant for $\sigma \preceq_k \tau$ never offers orchestration actions that do not correspond to actions offered by σ and that would never be enabled by τ . However, the existence of a relevant orchestrator is much less obvious than in the synchronous case. Indeed, it is clear that actions of the form $\langle \alpha, \bar{\alpha} \rangle$ and $\langle \varepsilon, \bar{\alpha} \rangle$ 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, $\bar{a}.e + \bar{b}.e + \bar{c}.a.e \dashv\vdash a \oplus b$ and $a \oplus b \preceq a$ and $\langle c, \varepsilon \rangle. \langle a, \bar{a} \rangle : \bar{a}.e + \bar{b}.e + \bar{c}.a.e \dashv\vdash a$. Simply removing the $\langle c, \varepsilon \rangle$ action (and the corresponding continuation) would produce the null orchestrator, which clearly cannot satisfy the client.

Proposition 7. Let $\sigma \preceq_k \tau$ and $\rho \dashv\vdash \sigma$. Then there exists a k -orchestrator g relevant for $\sigma \preceq_k \tau$ such that $g : \rho \dashv\vdash \tau$.

As for the synchronous case, when $\sigma \preceq_k \tau$ holds it is possible to find a universal k -orchestrator that makes every client that is strongly compliant with σ also weakly k -compliant with τ .

Proposition 8. $\sigma \preceq_k \tau$ if and only if there exists a k -orchestrator f such that $\rho \dashv\vdash \sigma$ implies $f : \rho \dashv\vdash \tau$ for every ρ .

Orchestrators as morphisms. Like synchronous orchestrators, asynchronous k -orchestrators can be seen as morphisms transforming service contracts. The exact definition of orchestration application is much more involved due to the fact that the orchestrator may synchronize with the service regardless of the behavior of the client.

Definition 20 (orchestrator application). The application of the orchestrator f to the (service) contract σ , notation $f(\sigma)$, is defined as

$$f(\sigma) \stackrel{\text{def}}{=} \bigoplus_{\sigma \dashv\vdash \mathbb{R}} \begin{cases} \sum_{f \xrightarrow{\langle \alpha, \varepsilon \rangle} f'} \alpha.f'(\sigma) + \sum_{f \xrightarrow{\langle \alpha, \bar{\alpha} \rangle} f', \alpha \in \mathbb{R}} \alpha.f'(\sigma(\alpha)) & \text{if } \{\alpha \mid \langle \varepsilon, \alpha \rangle \in \text{init}(f)\} \cap \bar{\mathbb{R}} = \emptyset \\ ((\sum_{f \xrightarrow{\langle \alpha, \varepsilon \rangle} f'} \alpha.f'(\sigma) + \sum_{f \xrightarrow{\langle \alpha, \bar{\alpha} \rangle} f', \alpha \in \mathbb{R}} \alpha.f'(\sigma(\alpha))) \oplus 0) & \\ + \bigoplus_{f \xrightarrow{\langle \varepsilon, \bar{\alpha} \rangle} f', \alpha \in \mathbb{R}} f'(\sigma(\alpha)) & \text{otherwise} \end{cases}$$

The equation reminds of the *expansion law* for the parallel operator in full CCS [21], 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 ($\{\alpha \mid \langle \varepsilon, \alpha \rangle \in \text{init}(f)\} \cap \bar{R} = \emptyset$): 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 ($\{\alpha \mid \langle \varepsilon, \alpha \rangle \in \text{init}(f)\} \cap \bar{R} \neq \emptyset$). 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 $f \stackrel{\text{def}}{=} \langle a, \varepsilon \rangle . \langle c, \varepsilon \rangle . (\langle \varepsilon, \bar{a} \rangle . \langle \bar{b}, b \rangle \vee \langle \varepsilon, \bar{c} \rangle . \langle \bar{d}, d \rangle)$. Then:

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

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

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 7. $f : \rho \dashv_k \sigma$ if and only if $\rho \dashv f(\sigma)$.

We are now able to connect the strong and weak k -subcontract relations as we did for the weak subcontract relation. The proof is identical to that of Corollary 1, except that Theorem 7 is used instead.

Corollary 2. $f : \sigma \preceq_k \tau$ if and only if $\sigma \sqsubseteq f(\tau)$.

The next few examples show that \preceq_k extends \preceq by permitting some permutation of actions:

- $\bar{a}.\bar{c}.b \preceq_1 \bar{c}.\bar{a}.b$ since $\bar{a}.\bar{c}.b = \langle \varepsilon, c \rangle . \langle \bar{a}, a \rangle . \langle \bar{c}, \varepsilon \rangle . \langle \bar{b}, b \rangle (\bar{c}.\bar{a}.b)$;
- $a.c.\bar{b} \preceq_1 c.a.\bar{b}$ since $a.c.\bar{b} = \langle a, \varepsilon \rangle . \langle c, \bar{c} \rangle . \langle \varepsilon, \bar{a} \rangle . \langle b, \bar{b} \rangle (c.a.\bar{b})$;
- $a.\bar{c}.\bar{b} \preceq_1 \bar{c}.a.\bar{b}$ since $a.\bar{c}.\bar{b} = \langle a, \varepsilon \rangle . \langle \bar{c}, c \rangle . \langle \varepsilon, \bar{a} \rangle . \langle b, \bar{b} \rangle (\bar{c}.a.\bar{b})$.

It is possible in general to postpone input actions. For instance we have $a.\beta.\sigma \preceq_k \beta.a.\sigma$ where the service on the r.h.s. of \preceq_k is able to perform the β action without having performed a first. On the other hand, we have $\bar{a}.b.\sigma \not\preceq_k b.\bar{a}.\sigma$ for every k because no valid orchestrator is capable of sending an \bar{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 \bar{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 9. *The following properties hold:*

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

Observe that, unlike the purely synchronous case, $f(\sigma) + f(\tau) \simeq f(\sigma + \tau)$ does not hold in general, because of the asynchronous actions that f may offer to the client side. Consider for example $f \stackrel{\text{def}}{=} \langle a, \varepsilon \rangle . (\langle \bar{b}, b \rangle + \langle \bar{d}, d \rangle)$. Then $f(\bar{b}) + f(\bar{d}) = a.\bar{b} + a.\bar{d} \simeq a.(\bar{b} \oplus \bar{d}) \sqsubseteq a.(\bar{b} + \bar{d}) = f(\bar{b} + \bar{d})$ but $f(\bar{b} + \bar{d}) \not\sqsubseteq f(\bar{b}) + f(\bar{d})$. Nonetheless Proposition 9 is still sufficient for proving that if $\sigma \sqsubseteq f(\sigma')$ and $\tau \sqsubseteq f(\tau')$, then $\sigma + \tau \sqsubseteq f(\sigma' + \tau')$ and $\sigma \oplus \tau \sqsubseteq f(\sigma' \oplus \tau')$.

Composing asynchronous orchestrators. As we have seen for synchronous orchestrators, monotonicity of orchestrator composition (Proposition 9) plays a central role in proving that \preceq_k is a pre-order. Unfortunately, a nasty consequence of buffering is that the functional composition of two orchestrator is not an orchestrator in general. To see why, consider for example

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

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

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

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 $\bar{b} \oplus \bar{d}$ in the resulting contract. No single orchestrator can turn $\bar{b} + \bar{d}$ into $a.c.(\bar{b} \oplus \bar{d})$ 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 .

The orchestrator composition operator \cdot is a generalization of \wedge that considers asynchronous orchestration actions:

Definition 21 (orchestrator composition). *The composition of two orchestrators f and g , notation $f \cdot g$, is defined as:*

$$f \cdot g \stackrel{\text{def}}{=} \bigvee_{f \xrightarrow{\langle \alpha, \varepsilon \rangle} f'} \langle \alpha, \varepsilon \rangle . (f' \cdot g) \vee \bigvee_{g \xrightarrow{\langle \varepsilon, \bar{\alpha} \rangle} g'} \langle \varepsilon, \bar{\alpha} \rangle . (f \cdot g')$$

$$\vee \bigvee_{f \xrightarrow{\langle \varphi, \bar{\alpha} \rangle} f', g \xrightarrow{\langle \alpha, \varphi' \rangle} g', \varphi \varphi' \neq \varepsilon} \langle \varphi, \varphi' \rangle . (f' \cdot g') \vee \bigvee_{f \xrightarrow{\langle \varepsilon, \bar{\alpha} \rangle} f', g \xrightarrow{\langle \alpha, \varepsilon \rangle} g'} (f' \cdot g')$$

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, $\langle \alpha, \bar{\alpha} \rangle \cdot \langle \alpha, \varepsilon \rangle = \langle \alpha, \varepsilon \rangle$ and $\langle \varepsilon, \bar{\alpha} \rangle \cdot \langle \alpha, \bar{\alpha} \rangle = \langle \varepsilon, \alpha \rangle$. 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 $f \cdot g = \langle a, \varepsilon \rangle \cdot \langle c, \varepsilon \rangle \cdot (\langle \bar{b}, b \rangle \vee \langle \bar{d}, d \rangle)$ and we observe $(f \cdot g)(\bar{b} + \bar{d}) = a.c.(\bar{b} + \bar{d})$.

The next result proves that $f \cdot g$ is correct and consequently that \preceq_k is a preorder:

Theorem 8. $f(g(\sigma)) \sqsubseteq (f \cdot g)(\sigma)$.

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 $(f \circ g)(\sigma) \not\sqsubseteq (f \cdot g)(\sigma)$ 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.

4.3 Alternative characterization of the weak k -subcontract relation

We now provide an alternative characterization of \preceq_k , which will also guide us in defining the algorithm for deciding \preceq_k in §7.³ As for synchronous orchestrators, asynchronous orchestrators modify the ready sets of the service as they are perceived by the client. The converse is also true, because of asynchronous orchestration actions. Assume A is a set of orchestration actions proposed by an asynchronous orchestrator. If R is a client ready set and S is a service ready set, then we write $A \circ S$ for the service ready set perceived by the client and we write $R \bullet A$ for the client ready set perceived by the service. These two modified ready sets can be defined thus:

$$\begin{aligned} A \circ S &\stackrel{\text{def}}{=} \{ \alpha \mid \langle \alpha, \varepsilon \rangle \in A \} \cup \{ \alpha \in S \mid \langle \alpha, \bar{\alpha} \rangle \in A \} \\ R \bullet A &\stackrel{\text{def}}{=} \{ \bar{\alpha} \mid \langle \varepsilon, \bar{\alpha} \rangle \in A \} \cup \{ \bar{\alpha} \in R \mid \langle \alpha, \bar{\alpha} \rangle \in A \} \end{aligned}$$

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

Definition 22 (coinductive weak k -subcontract). *We say that \mathcal{W}_k is a coinductive weak k -subcontract relation if $(\mathbb{B}, \sigma, \tau) \in \mathcal{W}_k$ implies that \mathbb{B} is a k -buffer and there exists a set of orchestration actions A such that $\mathbb{B} \vdash_k A$ and*

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

³ Unlike the strong and weak subcontract relations, a sound and complete deduction system for the weak k -subcontract relation is not known at the time of this writing.

Condition (1) requires that either τ can be made more deterministic than σ by means of the orchestrator (the ready set $A \circ S$ 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 ($(\emptyset \bullet A) \cap \bar{S} \neq \emptyset$ implies that $\langle \varepsilon, \bar{\alpha} \rangle \in A$ and $\alpha \in S$ 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 9. \preceq_k is the largest coinductive weak k -subcontract relation.

5 Contract duality with orchestration

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

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

Definition 23 (viable contract). A (client) contract ρ is viable, notation $\text{viable}(\rho)$, if there exists σ such that $\rho \dashv \sigma$.

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

Definition 24 (coinductive viability). We say that the predicate \mathcal{V} is a coinductive viability if $\rho \in \mathcal{V}$ and $\rho \Downarrow R$ implies either $e \in R$ or $\rho(\alpha) \in \mathcal{V}$ for some $\alpha \in R$.

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

Proposition 10. $\text{viable}(\cdot)$ is the largest coinductive viability.

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

Definition 25 (dual contract). Let ρ be a viable client contract. The dual contract of ρ , denoted by ρ^\perp , is defined as:

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

The idea of the dual operator is to consider every state R of the client in which the client cannot terminate successfully ($e \notin R$). 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 \bar{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:

- $(a.e)^\perp = (a.e \oplus e)^\perp = \bar{a}$ (the service must provide \bar{a});
- $(a.e + e)^\perp = 0$ (the service need not provide anything because the client can terminate immediately);
- $(a.e + b.e)^\perp = \bar{a} \oplus \bar{b}$ (the service can decide whether to provide \bar{a} or \bar{b});
- $(a.e \oplus b.e)^\perp = \bar{a} + \bar{b}$ (the service must provide both \bar{a} and \bar{b}).

Theorem 10 (duality). *Let ρ be a viable client contract. Then*

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

The assumption of using orchestrators is essential as far as duality is concerned: $(a.e + e)^\perp = 0$ but 0 is *not* the smallest (according to \sqsubseteq) contract satisfying $a.e + e$. For example, $0 \oplus b \sqsubseteq 0$ and $a.e + e \dashv 0 \oplus b$. On the contrary, 0 is the least element of \preceq_k 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 [24].

6 Contracts for infinite behaviors

Until now we have been working with *finite* contracts, namely with contracts that provide a maximum number of nested prefixes. This means that we can only model client and service behaviors corresponding to *finite* interactions, but we cannot model, for example, a client and a service that interact infinitely often if the client performs an unbounded number of requests and the service is able to satisfy them all.

There are three well-known ways of dealing with repeated (or recursive) behavior. In regular expressions, for example, the algebra includes a $*$ operator (the so-called Kleene star) so that, for example, a^* denotes the set of arbitrarily long, but finite, sequences of a symbols. In process algebra, it is more frequent the use of *definitions* or of *recursion terms*. In the former case, the syntax is extended to include variables X , and a global environment defines a finite set of equations of the form

$$X \stackrel{\text{def}}{=} \sigma$$

giving the meaning to variables. In the latter case, the syntax is extended with two constructs: recursion is expressed by a term $\text{rec } x.\sigma$ which binds the recursion variable x within σ ; every occurrence of the variable x within σ stands for an occurrence of the whole term $\text{rec } x.\sigma$. In other words, the contracts $\text{rec } x.\sigma$ and its *unfolding* $\sigma\{\text{rec } x.\sigma/x\}$

(the latter being the same as σ except that every free occurrence of x has been replaced by $\text{rec } x.\sigma$) are equivalent.

Observe that all the three approaches require an extension of the syntax of contracts (Definition 1) as well as corresponding extensions in their operational semantics (Definition 2). In addition, it is not obvious how to apply the approach using the Kleene star in our scenario since it is well known [17, 2] that, in a nondeterministic setting, there are behaviors that cannot be expressed using that syntax.

Instead, we resort to the theory of regular trees [14]. This approach is more commonly used in type theory and has the remarkable advantage that it virtually requires no change at all to the whole theory we have developed so far. The basic idea is extremely simple: infinite contracts are represented as infinite terms generated by the grammar in Definition 1. In practice, this amounts to working directly with the solutions of the equations $X \stackrel{\text{def}}{=} \sigma$ or with the infinite unfoldings of recursive terms $\text{rec } x.\sigma$. However, saying that we use “infinite terms” is not enough to ensure that such terms do actually make sense or that the theory remains decidable. To guarantee these facts we must enforce two additional constraints, hence the precise definition of the infinite terms we consider is the following:

Definition 26 (contracts). *The set of contracts is the set of possibly infinite, regular trees generated by the grammar in Definition 1 such that every infinite branch of a tree contains infinitely many occurrences of “.” (the prefix constructor).*

This definition imposes two fundamental requirements on the infinite terms we have mentioned earlier. First of all, we require that the term must be a *regular tree*. A regular tree contains finitely many different subtrees (observe that every finite term is a regular tree). Second, we impose a *contractivity condition*⁴ by requiring that if a contract term is infinite, it is because it proposes an unbound sequence of actions. For example, we exclude regular terms that are solutions of the equations $X = X + X$ or $X = X \oplus X$. As a side effect of the contractivity condition, we enforce the fact that the behaviors we consider are *everywhere convergent*. Namely, contracts can never exhibit an infinite number of internal transitions, hence we are not able to describe the behavior of clients and services that may diverge. This is in practice the only limitation of our approach, if compared to the more traditional approaches that deal with infinite behaviors. The reader interested in one possible way of dealing with divergent contracts can refer to [24].

The definition of orchestrators can be extended to infinite behaviors in a similar way, by imposing regularity and contractivity of the orchestrator.

We now revisit the definitions and results we have developed for the theory of finite contracts and see now they change when we consider contracts as by Definition 26.

The strong compliance relation (Definition 3) now takes into account the possibility that client and service interact infinitely often. Observe that, since contracts are everywhere convergent, every infinite computation starting from $\rho \parallel \sigma$ involves an infinite number of synchronizations between client and service. For example, we have $\bar{a}.a.\bar{a}.\dots \dashv a.a.a.\dots$.

⁴ The term “contractivity” has nothing to do with the fact that we talk about “contracts”, see [14].

As regards contract continuations (Definition 5), observe that $\sigma(\alpha)$ is still well defined because contracts are everywhere convergent: there is always a finite number of residuals σ' such that $\sigma \xRightarrow{\alpha} \sigma'$. In fact we can state an even stronger property which is fundamental for asserting the well-foundedness of several definitions.

Proposition 11. *Let $D(\sigma) \stackrel{\text{def}}{=} \{\sigma(\varphi) \mid \sigma \xRightarrow{\varphi}\}$. Then $D(\sigma)$ is finite for every σ .*

Proof. It is sufficient to observe that $\sigma(\varphi)$ is the internal choice of some subtrees 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 [25]. \square

A similar result can be proved for orchestrators and the corresponding $\xrightarrow{\mu}$ relation. Because of these results, it is easy to see that all the inductive definitions given in the paper are well founded, not in the sense that a base case is eventually reached, but that they yield regular and contractive contracts and orchestrators. Consider for example the definition of the identity orchestrator $I(\sigma)$ for a contract σ . According to equation 3 on page 13 we see that $I(\sigma)$ depends on $I(\sigma(\alpha))$ for a finite set of actions α . In turn, each $I(\sigma(\alpha))$ will depend upon a finite set of $I(\sigma(\alpha)(\beta))$. Now observe that $\sigma(\alpha)(\beta) = \sigma(\alpha\beta)$. Hence, if we think of $I(\sigma(\varphi))$ as of a label, Proposition 11 let us conclude that $I(\sigma)$ depends on a finite number of labels $I(\sigma(\varphi))$, hence it can be represented as a regular orchestrator. A similar reasoning can be applied for deducing that the definitions of orchestrator composition, orchestrator application (Definition 14), asynchronous orchestrator application (Definition 20), and other inductive definitions are all well founded.

As regards the deduction system presented in §3.4 for the weak subcontract relation, it is still complete, provided that we admit infinite (but regular and contractive) proofs, where a contractive proof is one where every infinite branch of the derivation tree contains infinite occurrences of the (PREFIX) rule. Details can be found in the full version of [10]. Observe that if divergence is added to the contract language, the deduction system must be suitably extended with new axioms [21].

As regards asynchronous orchestrators, the validity constraint and the finiteness of the rank have fundamental consequences: a valid orchestrator of finite rank cannot read an unbound number of messages from just one of the two interacting parties. Thus, finiteness of the orchestrator's rank guarantees that the orchestrator is *fair*: it still holds, as in the case of synchronous orchestrators, that every infinite interaction between a client and a service is such that both client and service interact infinitely often.

Finally, the proofs of all the results presented are all still valid, since none of them, except for the completeness proof of the deduction system, proceed by induction on the depth of contracts or orchestrators (as a matter of facts, most proofs are based on coinduction).

7 Automatic synthesis of orchestrators

In this section we devise an algorithm for computing the k -orchestrator witnessing $\sigma \preceq_k \tau$, provided there is one. Actually, we have already seen that there can be more

than one orchestrator proving a relation $\sigma \preceq_k \tau$, 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 $f : \sigma \preceq_k \tau$ and $g : \sigma \preceq_k \tau$, then $f \vee g : \sigma \preceq_k \tau$. So we may naively attempt to define an algorithm that synthesizes the *largest* orchestrator, according to their trace semantics. This approach is not effective since the largest orchestrator proving $\sigma \preceq_k \tau$ involves an infinite number of different names, and thus is not representable as a proper orchestrator. The idea is that, by means of Proposition 7, we may restrict our interest to the subclass of the orchestrators that are relevant for $\sigma \preceq_k \tau$.

Definition 27 (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 $g \leq f$.*

The algorithm that synthesizes the best relevant orchestrator proving $\sigma \preceq_k \tau$, provided there is one, is defined inductively by the rule:

$$\frac{\begin{array}{l} A_r = \{ \langle \varphi, \bar{\varphi}' \rangle \mid \sigma \xrightarrow{\varphi}, \tau \xrightarrow{\varphi'}, \mathbb{B} \vdash_k \langle \varphi, \bar{\varphi}' \rangle \} \\ A = \{ \langle \varphi, \bar{\varphi}' \rangle \in A_r \mid \mathbb{B} \langle \varphi, \bar{\varphi}' \rangle \vdash_k f_{\langle \varphi, \bar{\varphi}' \rangle} : \sigma(\varphi) \preceq \tau(\varphi') \} \\ \tau \Downarrow S \Rightarrow (\exists R : \sigma \Downarrow R \wedge R \subseteq A \circ S) \vee (\emptyset \bullet A) \cap \bar{S} \neq \emptyset \end{array}}{\mathbb{B} \vdash_k \bigvee_{\mu \in A} \mu \cdot f_\mu : \sigma \preceq \tau}$$

A judgment of the form $\mathbb{B} \vdash_k f : \sigma \preceq \tau$ means that f is a k -orchestrator proving that $\sigma \preceq_k \tau$ when the buffer of the orchestrator is in state \mathbb{B} . 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 $f : \sigma \preceq_k \tau$ if $\emptyset \vdash_k f : \sigma \preceq \tau$.

Although the algorithm looks formidable, it embeds the conditions in Definition 7 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 $\langle \bar{a}, \varepsilon \rangle$ is enabled only if $\mathbb{B}(\circ, \bar{a}) > 0$; an action $\langle a, \varepsilon \rangle$ is enabled only if the buffer has not reached its capacity, namely if $\mathbb{B}(\bullet, \bar{a}) < k$; 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 $\langle \varphi, \bar{\varphi}' \rangle \in A$ we are able to find an orchestrator $f_{\langle \varphi, \bar{\varphi}' \rangle}$ that proves $\tau(\varphi') \preceq_k \sigma(\varphi)$ (second premise of the rule). When checking the continuations, 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 15, 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 .

Observe that the algorithm hides a base case, when the set A_r is empty. This case is eventually reached when σ and τ are *finite* contracts, since each application of the rule

in the algorithm decreases the *depth* of either σ or τ , or both.⁵ The algorithm can also be extended for dealing with possibly infinite (but regular) contracts, as by Definition 26, using standard memoization techniques. The reader interested in the details can refer to [27].

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

Theorem 11. *The following properties hold:*

1. (termination) it is decidable to check whether there exists f such that $f : \sigma \triangleleft_k \tau$;
2. (correctness) $f : \sigma \triangleleft_k \tau$ 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 $g : \sigma \triangleleft_k \tau$ for some g such that $f \leq g$.

8 Application example

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

$$P_i \stackrel{\text{def}}{=} \text{fork}_i.\text{fork}_i.\overline{\text{thought}}.\overline{\text{fork}}.\overline{\text{fork}}$$

where the fork_i actions model the philosopher's request of two forks, $\overline{\text{thought}}$ models the generation of a thought, and the $\overline{\text{fork}}$ 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.

We consider two potential clients of such services, a “sloppy” client C and a “diligent” client D which are defined as follows:

$$\begin{aligned} C &\stackrel{\text{def}}{=} \sum_{i=1..2} \overline{\text{fork}}_i.\sum_{i=1..2} \overline{\text{fork}}_i.\overline{\text{thought}}.\overline{\text{fork}}.\overline{\text{fork}} \\ D &\stackrel{\text{def}}{=} \sum_{i=1..2} \overline{\text{fork}}_i.\overline{\text{fork}}_i.\overline{\text{thought}}.\overline{\text{fork}}.\overline{\text{fork}} \end{aligned}$$

The difference between the two clients is that the sloppy one provides two forks, but it does not care that the two forks it provides end up to the same philosopher. In a system with two philosophers this may cause the system to deadlock. The diligent client provides two forks and it does not care which philosopher gets the forks, but it does care that the two forks end up to the same philosopher.

In order to see which services can satisfy these clients, we compute the corresponding dual contracts and we obtain:

$$\begin{aligned} C^\perp &\stackrel{\text{def}}{=} \bigoplus_{i=1..2} \text{fork}_i.\bigoplus_{i=1..2} \text{fork}_i.\overline{\text{thought}}.\overline{\text{fork}}.\overline{\text{fork}} \\ D^\perp &\stackrel{\text{def}}{=} \bigoplus_{i=1..2} \text{fork}_i.\overline{\text{fork}}_i.\overline{\text{thought}}.\overline{\text{fork}}.\overline{\text{fork}} \end{aligned}$$

We observe that $C^\perp \sqsubseteq P_i$ and $D^\perp \sqsubseteq P_i$ for $i = 1..2$, hence we conclude that $C \dashv P_i$ and $D \dashv P_i$. Namely, both clients are satisfied by any service that hires one of the two

⁵ The depth of a finite contract is the maximum number of nested prefixes in it.

philosophers for generating thoughts. Consider now a richer service provider who can afford two hire *both* philosophers. Intuitively, this service has contract $P_1 | P_2$, which denotes the *parallel composition* of P_1 and P_2 . Even though our contract language is not equipped with a parallel composition operator, we can faithfully capture its semantics using just the operators provided by our contract language. Assuming that σ and τ are the contracts of two services that never synchronize with each other, which is the case in this example, we can express $\sigma | \tau$ using a simplified form of *expansion law* [21]:

$$\sigma | \tau \stackrel{\text{def}}{=} \bigoplus_{\sigma \downarrow \mathbb{R}, \tau \downarrow \mathbb{S}} (\sum_{\alpha \in \mathbb{R}} \alpha.(\sigma(\alpha) | \tau) + \sum_{\alpha \in \mathbb{S}} \alpha.(\sigma | \tau(\alpha)))$$

With this definition, we can now see that $C^\perp \not\sqsubseteq P_1 | P_2$ and $D^\perp \not\sqsubseteq P_1 | P_2$, hence the richer service with contract $P_1 | P_2$ cannot be used as is to satisfy either client. Let us now run the algorithm to see whether $C^\perp \preceq_0 P_1 | P_2$. If we consider the sequence of actions $\overline{\text{fork}}_1 \overline{\text{fork}}_2$ we reduce to checking

$$\overline{\text{thought}}.\overline{\text{fork}}.\overline{\text{fork}} \preceq_0 P_1(\overline{\text{fork}}_1) | P_2(\overline{\text{fork}}_2)$$

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

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

which allows us to prove $f : C^\perp \preceq_0 P_1 | P_2$.

In this particular example, the same orchestrator f also allows us to prove $f : D^\perp \preceq_0 P_1 | P_2$. However, there is an important difference with the case of C^\perp because we have $I(D^\perp) \leq f$. This means that D^\perp never exposes any action that must be filtered out by f . Since D^\perp is the dual of D , this in turn means that D never tries to perform any action that is not filtered by f . In conclusion, while $D^\perp \not\sqsubseteq P_1 | P_2$, hence there are clients of D^\perp that are not satisfied by $P_1 | P_2$ if not by means of some orchestrator, it is the case that $D \dashv P_1 | P_2$ and we are able to conclude this fact from $f : D^\perp \preceq_0 P_1 | P_2$ and $I(D^\perp) \leq f$.

Suppose now that the rich 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

$$Q_i \stackrel{\text{def}}{=} \overline{\text{fork}}_i.\overline{\text{fork}}_i.\overline{\text{fork}}.\overline{\text{fork}}.\overline{\text{thought}}$$

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 $P_1 | P_2 \preceq_k Q_1 | Q_2$ for some k and the interesting step is when the algorithm eventually checks $P_1(\overline{\text{fork}}_1 \overline{\text{fork}}_1) | P_2 \preceq_k Q_1(\overline{\text{fork}}_1 \overline{\text{fork}}_1) | Q_2$ (symmetrically for P_2 and the sequence of actions $\overline{\text{fork}}_2 \overline{\text{fork}}_2$). At this stage $P_1(\overline{\text{fork}}_1 \overline{\text{fork}}_1) | P_2$ has just the ready set $\{\overline{\text{thought}}, \overline{\text{fork}}_2\}$, whereas the contract $Q_1(\overline{\text{fork}}_1 \overline{\text{fork}}_1) | Q_2$ has just the ready set $\{\overline{\text{fork}}, \overline{\text{fork}}_2\}$. By accepting the two $\overline{\text{fork}}$ messages asynchronously we reduce to checking whether $P_1(\overline{\text{fork}}_1 \overline{\text{fork}}_1) |$

$P_2 \preceq_k \overline{\text{thought}}.Q_1 \mid Q_2$, which holds by allowing the $\overline{\text{thought}}$ action to occur, followed by the asynchronous sending of the two buffered $\overline{\text{fork}}$ messages. Overall the relation is proved by the orchestrator

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

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

$$f \cdot g = \bigvee_{i=1..2} \langle \overline{\text{fork}}_i, \overline{\text{fork}}_i \rangle. \langle \overline{\text{fork}}_i, \overline{\text{fork}}_i \rangle. \langle \varepsilon, \overline{\text{fork}} \rangle. \langle \varepsilon, \overline{\text{fork}} \rangle. \langle \overline{\text{thought}}, \overline{\text{thought}} \rangle. \langle \overline{\text{fork}}, \varepsilon \rangle$$

9 Related and future work

This work originated by revisiting CCS without τ 's [16] in the context of Web services. Contracts are in fact just a concrete representation of *acceptance trees* [20, 21]. Early attempts to define a reasonable subcontract relation [9] have eventually led to the conclusion that some control over actions is necessary: [24] proposes a static form of control that makes use of explicit contract interfaces whereas [10] proposes a dynamic form of control by means of so-called *filters* and [27] elaborates on the idea of [10] 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.

[18] 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 [9] and that can be traced back to the LOTOS language [7]. The authors of [18] 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. Thus, the work done in [10, 27] 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.

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 [26] 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 (§4).

There has been extensive research on the automatic synthesis of connectors both in the domain of software architectures (see for example [23]) and also in the more specific domain of Web services [30, 5, 22, 28, 19]. 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.

Future work. The presented theory of contracts is based on (a variant of) the CCS language, whose terms describe interactions without any information on the content of the exchanged messages. Consequently, this theory is adequate as long as the topology of the interaction network and the roles (client/service) of the participants to an interaction are fixed. In more complex scenarios, one can be interested in modelling systems with multiple participants that interact by opening private sessions and by exchanging (delegating) opened sessions between them. In this scenario, the crisp distinction between client and service is lost, and the exchange of communication channels during interaction cannot be adequately captured by a CCS-based contract language. Consequently, the theory presented in this paper is currently being extended to a π -calculus based contract language, which enables the description of processes creating and exchanging communication channels.

Acknowledgments. Parts of the work presented in this paper have been developed in collaboration with Samuele Carpineti, Giuseppe Castagna, Nils Gesbert, and Cosimo Laneve. The sound and complete deduction system in §3 is an adaptation of the one in the full version of [10] which, in turn, closely follows the one for the must preorder presented in [21].

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. Ed Brinksma, Giuseppe Scollo, and Chris Steenbergen. *LOTOS specifications, their implementations and their tests*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.
8. Kim B. Bruce and Giuseppe Longo. A modest model of records, inheritance and bounded quantification. *Information and Computation*, 87(1/2):196–240, 1990.
9. 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.
10. Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A theory of contracts for Web services. In *Proceedings of POPL'08*, pages 261–272. ACM, 2008.
11. Gang Chen. Soundness of coercion in the calculus of constructions. *Journal of Logic and Computation*, 14(3):405–427, 2004.
12. Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, 2007.
13. John Colgrave and Karsten Januszewski. Using WSDL in a UDDI registry, version 2.0.2. Technical note, OASIS, 2004.
14. Bruno Courcelle. Fundamental properties of infinite trees. *Theoretical Computer Science*, 25:95–169, 1983.
15. Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
16. Rocco De Nicola and Matthew Hennessy. CCS without τ 's. In *TAPSOFT'87/CAAP'87*, number 249 in LNCS, pages 138–152. Springer, 1987.
17. Rocco De Nicola and Anna Labella. Nondeterministic regular expressions as solutions of equational systems. *Theor. Comput. Sci.*, 302(1-3):179–189, 2003.
18. Rik Eshuis and Maarten M. Fokkinga. Comparing refinements for failure and bisimulation semantics. *Fundamenta Informaticae*, 52(4):297–321, 2002.
19. Giuseppe De Giacomo and Sebastian Sardiña. Automatic synthesis of new behaviors from a library of available behaviors. In *IJCAI*, pages 1866–1871, 2007.
20. Matthew Hennessy. Acceptance trees. *JACM: Journal of the ACM*, 32(4):896–928, 1985.
21. Matthew Hennessy. *Algebraic Theory of Processes*. Foundation of Computing. MIT Press, 1988.
22. 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.
23. Paola Inverardi and Massimo Tivoli. Software architecture for correct components assembly. In *SFM'03*, pages 92–121, 2003.
24. 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.

25. 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.
26. Jayadev Misra and William R. Cook. Computation orchestration – a basis for wide-area computing. *Software and Systems Modeling*, 6(1):83–0110, 2007.
27. Luca Padovani. Contract-directed synthesis of simple orchestrators. In *CONCUR'08*, number 5201 in LNCS, pages 131–146. Springer, 2008.
28. 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.
29. Sergei Soloviev, Alex Jones, and Zhaohui Luo. Some Algorithmic and Proof-Theoretical Aspects of Coercive Subtyping. In *TYPES'96*. LNCS 1512, 173–196, Springer, 1996.
30. Paolo Traverso and Marco Pistore. Automated composition of semantic web services into executable processes. In *International Semantic Web Conference*, pages 380–394, 2004.
31. Claus von Riegen and Ivana Trickovic. Using BPEL4WS in a UDDI registry. Technical note, OASIS, 2004.

A Proofs

A.1 Proofs of §2

Proof (of Theorem 1). First of all we prove that \sqsubseteq is a coinductive strong subcontract relation. Let $\sigma \sqsubseteq \tau$. As regards condition (1) in Definition 7, let $\{R_1, \dots, R_n\}$ be the ready sets of σ and assume by contradiction that there exists S such that $\tau \Downarrow S$ and $R_i \not\subseteq S$ for every $1 \leq i \leq n$. Namely, for every $1 \leq i \leq n$ there exists $\alpha_i \in R_i \setminus S$. Consider $\rho \stackrel{\text{def}}{=} \sum_{1 \leq i \leq n} \bar{\alpha}_i.e$. Then $\rho \dashv \sigma$ but $\rho \not\vdash \tau$, which is absurd by hypothesis. As regards condition (2), let $\tau \xrightarrow{\alpha}$ and assume by contradiction that $\sigma \not\xrightarrow{\alpha}$. Consider $\rho \stackrel{\text{def}}{=} e + \bar{\alpha}$. Then $\rho \dashv \sigma$ but $\rho \not\vdash \tau$, which is absurd by hypothesis. Let ρ' be a client contract such that $\rho' \dashv \sigma(\alpha)$ and consider $\rho \stackrel{\text{def}}{=} e + \bar{\alpha}.\rho'$. 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 \mathcal{S} is included in \sqsubseteq . Let $(\sigma, \tau) \in \mathcal{S}$ and assume $\rho \dashv \sigma$. Consider now a maximal computation $\rho \mid \tau \Longrightarrow \rho' \mid \tau' \dashv \dashv$. We can “unzip” this derivation into two derivations $\rho \xrightarrow{\varphi} \rho' \dashv \dashv$ and $\tau \xrightarrow{\varphi} \tau' \dashv \dashv$ for some string φ of actions. From condition (2) of Definition 7 and by induction on φ we derive that $\sigma \xrightarrow{\varphi}$ and $(\sigma(\varphi), \tau(\varphi)) \in \mathcal{S}$. From $\tau(\varphi) \Downarrow \text{init}(\tau')$ and condition (1) of Definition 7 we derive that there exists $R \subseteq \text{init}(\tau')$ such that $\sigma(\varphi) \Downarrow R$. By definition of ready set we obtain that there exists σ' such that $\sigma \xrightarrow{\varphi} \sigma' \dashv \dashv$ and $\text{init}(\sigma') \subseteq \text{init}(\tau')$. We can now “zip” the two derivations starting from ρ and σ and obtain a derivation $\rho \mid \sigma \Longrightarrow \rho' \mid \sigma'$. We observe that $\rho' \mid \sigma' \dashv \dashv$ since $\rho' \dashv \dashv$ and $\sigma' \dashv \dashv$ and $\text{init}(\sigma') \subseteq \text{init}(\tau')$. From $\rho \dashv \sigma$ we conclude $\rho' \xrightarrow{e}$. \square

The proof of Proposition 1 is omitted. A proof of property (1) can be found in [24], while a proof that \sqsubseteq is a precongruence with respect to the operators of the contract language is included in [21].

A.2 Proofs of §3

Proof (of Proposition 2). 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 σ^\top , can be defined thus:

$$\sigma^\top \stackrel{\text{def}}{=} \sum_{\sigma \downarrow_R} \begin{cases} e & \text{if } R = \emptyset \\ \bigoplus_{\alpha \in R} \bar{\alpha} \cdot \sigma(\alpha)^\top & \text{otherwise} \end{cases}$$

It is trivial to verify that $\sigma^\top \dashv \sigma$. Let f be an orchestrator such that $f : \sigma^\top \dashv \tau$ and assume, without loss of generality, that f is relevant for $\sigma \preceq \tau$. We prove that $f : \sigma \preceq \tau$ by contradiction. Assume that there exists ρ such that $\rho \dashv \sigma$ and $f : \rho \not\vdash \tau$. Then there exists a derivation $\rho \parallel_f \tau \Longrightarrow \rho' \parallel_{f'} \tau' \dashv \dashv$ such that $\rho' \xrightarrow{e}$. Consequently there exist $\alpha_1, \dots, \alpha_n$ and such that $\rho \xrightarrow{\bar{\alpha}_1 \cdots \bar{\alpha}_n} \rho' \dashv \dashv$ and $\tau \xrightarrow{\alpha_1 \cdots \alpha_n} \tau' \dashv \dashv$ and $f \vdash \langle \alpha_1, \bar{\alpha}_1 \rangle \cdots \langle \alpha_n, \bar{\alpha}_n \rangle \rightarrow f'$. Since f is relevant, it is easy to deduce, by induction on n , that $\sigma \xrightarrow{\alpha_1 \cdots \alpha_n}$. From $\rho' \parallel_{f'} \tau' \dashv \dashv$ we deduce that $\rho' \xrightarrow{\bar{\alpha}}$ and $f' \vdash \langle \alpha, \bar{\alpha} \rangle$ imply $\tau' \xrightarrow{\alpha}$. Let R_1, \dots, R_m be the ready sets of $\sigma(\alpha_1 \cdots \alpha_n)$. From $\rho \dashv \sigma$ and $\rho' \xrightarrow{e}$ we deduce that for every $1 \leq i \leq n$ there exists $\beta_i \in R_i$ and $\rho' \xrightarrow{\bar{\beta}_i}$. By definition of most demanding client we have $\sigma(\alpha_1 \cdots \alpha_n)^\top \downarrow \{\bar{\beta}_1, \dots, \bar{\beta}_m\}$, because each ready set of $\sigma(\alpha_1 \cdots \alpha_n)^\top$ is obtained by taking one action from every non-empty ready set of $\sigma(\alpha_1 \cdots \alpha_n)$. Hence there exists ρ'' such that $\sigma^\top \xrightarrow{\bar{\alpha}_1 \cdots \bar{\alpha}_n} \rho'' \dashv \dashv$ and $\text{init}(\rho'') = \{\bar{\beta}_1, \dots, \bar{\beta}_m\} \subseteq \text{init}(\rho')$. By zipping the derivations starting from σ^\top , f , and τ we obtain a derivation $\sigma^\top \parallel_f \tau \Longrightarrow \rho'' \parallel_{f'} \tau' \dashv \dashv$ where $\rho'' \xrightarrow{e}$. This is absurd, for $f : \sigma^\top \dashv \tau$ by hypothesis. \square

Proof (of Theorem 3). (\Rightarrow) Assume $f : \rho \dashv \sigma$ and consider a derivation $\rho \parallel f(\sigma) \Longrightarrow \rho' \parallel \tau \dashv \dashv$. Then there exist φ and f' such that $\rho \xrightarrow{\bar{\varphi}} \rho' \dashv \dashv$ and $f(\sigma) \xrightarrow{\varphi} f'(\sigma(\varphi)) \Longrightarrow \tau \dashv \dashv$. By definition of $f(\sigma)$ there exist $\alpha_1, \dots, \alpha_n$ and such that $\varphi = \alpha_1 \cdots \alpha_n$ and $f \vdash \langle \alpha_1, \bar{\alpha}_1 \rangle \cdots \langle \alpha_n, \bar{\alpha}_n \rangle \rightarrow f'$. From $\rho' \parallel \tau \dashv \dashv$ we deduce $\overline{\text{init}(\rho')} \cap \text{init}(\tau) = \emptyset$. From $f'(\sigma(\varphi)) \Longrightarrow \tau \dashv \dashv$ and by definition of $f'(\sigma(\varphi))$ we deduce that there exists σ' such that $\sigma \xrightarrow{\varphi} \sigma' \dashv \dashv$ and $\text{init}(f'(\sigma')) = \text{init}(\tau)$. By zipping the derivations starting from ρ , f , and σ we obtain $\rho \parallel_f \sigma \Longrightarrow \rho' \parallel_{f'} \sigma' \dashv \dashv$. Furthermore $\rho' \parallel_{f'} \sigma' \dashv \dashv$ because $\rho' \dashv \dashv$ and $\sigma' \dashv \dashv$. From the hypothesis $f : \rho \dashv \sigma$ we conclude $\rho' \xrightarrow{e}$.

(\Leftarrow) Assume $\rho \dashv f(\sigma)$ and consider a derivation $\rho \parallel_f \sigma \Longrightarrow \rho' \parallel_{f'} \sigma' \dashv \dashv$. By “un-zipping” this derivation we have that there exist $\alpha_1, \dots, \alpha_n$ such that $\rho \xrightarrow{\bar{\alpha}_1 \cdots \bar{\alpha}_n} \rho' \dashv \dashv$ and $f \vdash \langle \alpha_1, \bar{\alpha}_1 \rangle \cdots \langle \alpha_n, \bar{\alpha}_n \rangle \rightarrow f'$ and $\sigma \xrightarrow{\alpha_1 \cdots \alpha_n} \sigma' \dashv \dashv$. Furthermore from $\rho' \parallel_{f'} \sigma' \dashv \dashv$ we derive $\overline{\text{init}(\rho')} \cap \text{init}(f'(\sigma')) = \emptyset$. By definition of $f(\sigma)$ there exists τ such that $f(\sigma) \xrightarrow{\alpha_1 \cdots \alpha_n} \tau \dashv \dashv$ and $\text{init}(\tau) = \text{init}(f'(\sigma'))$. By zipping the derivations starting from ρ and $f(\sigma)$ we obtain $\rho \parallel f(\sigma) \Longrightarrow \rho' \parallel \tau$. Furthermore $\rho' \parallel \tau \dashv \dashv$ because $\rho' \dashv \dashv$, $\tau \dashv \dashv$, and $\overline{\text{init}(\rho')} \cap \text{init}(\tau) = \overline{\text{init}(\rho')} \cap \text{init}(f'(\sigma')) = \emptyset$. From $\rho \dashv f(\sigma)$ we conclude $\rho' \xrightarrow{e}$. \square

Proof (of Proposition 3). We prove item (1); items (2) and (3) are similar. By Theorem 3 it is sufficient to prove that $f : \rho \dashv \sigma$ implies $f : \rho \dashv \tau$ for every ρ , under

the hypothesis $\sigma \sqsubseteq \tau$. Consider a derivation $\rho \parallel_f \tau \Longrightarrow \rho' \parallel_{f'} \tau' \dashrightarrow$. Then there exist $\alpha_1, \dots, \alpha_n$ such that $\rho \xrightarrow{\bar{\alpha}_1 \dots \bar{\alpha}_n} \rho' \dashrightarrow$ and $f \xrightarrow{\langle \alpha_1, \bar{\alpha}_1 \rangle \dots \langle \alpha_n, \bar{\alpha}_n \rangle} f'$ and $\tau \xrightarrow{\alpha_1 \dots \alpha_n} \tau' \dashrightarrow$. Furthermore, from $\rho' \parallel_{f'} \tau' \dashrightarrow$ we deduce $\overline{\text{init}(\rho')} \cap \text{init}(f'(\tau')) = \emptyset$. From $\sigma \sqsubseteq \tau$ we derive that there exists σ' such that $\sigma \xrightarrow{\alpha_1 \dots \alpha_n} \sigma' \dashrightarrow$ and $\text{init}(\sigma') \subseteq \text{init}(\tau')$. By zipping the derivations starting from ρ, f , and σ we obtain $\rho \parallel_f \sigma \Longrightarrow \rho' \parallel_{f'} \sigma'$. Furthermore, $\overline{\text{init}(\rho')} \cap \text{init}(f'(\sigma')) \subseteq \overline{\text{init}(\rho')} \cap \text{init}(f'(\tau')) = \emptyset$, hence $\rho' \parallel_{f'} \sigma' \dashrightarrow$. From $f : \rho \dashv\vdash \sigma$ we conclude $\rho' \xrightarrow{e}$. \square

Proof (of Theorem 4). First we show that \preceq is a coinductive weak subcontract relation. Let $f : \sigma \preceq \tau$ and assume, without loss of generality, that f is relevant for $\sigma \preceq \tau$. It is sufficient to prove that

$$\mathscr{W} \stackrel{\text{def}}{=} \{(\sigma(\alpha_1 \dots \alpha_n), \tau(\alpha_1 \dots \alpha_n)) \mid f \xrightarrow{\langle \alpha_1, \bar{\alpha}_1 \rangle \dots \langle \alpha_n, \bar{\alpha}_n \rangle}\}$$

is a coinductive weak subcontract relation. Let $(\sigma', \tau') \in \mathscr{W}$. Then there exist $\alpha_1, \dots, \alpha_n$ and $\alpha_1, \dots, \alpha_n$ and f' such that $f \xrightarrow{\langle \alpha_1, \bar{\alpha}_1 \rangle \dots \langle \alpha_n, \bar{\alpha}_n \rangle} f'$, $\sigma' = \sigma(\alpha_1 \dots \alpha_n)$, and $\tau' = \tau(\alpha_1 \dots \alpha_n)$. Let $A \stackrel{\text{def}}{=} \text{init}(f')$. As regards condition (1) of Definition 15, let R_1, \dots, R_m be the ready sets of σ' . Assume by contradiction that there exists S such that $\tau' \Downarrow S$ and $R_i \not\subseteq A \circ S$ for every $1 \leq i \leq m$. Then there exists $\alpha_i \in R_i \setminus A \circ S$ for every $1 \leq i \leq m$. Let $\rho \stackrel{\text{def}}{=} \sum_{1 \leq i \leq m} \bar{\alpha}_i.e$. We have $\rho \dashv\vdash \sigma'$ but $f' : \rho \not\vdash \tau'$, which is absurd. As regards condition (2) of Definition 15, assume $\tau' \xrightarrow{\alpha}$ and $\langle \alpha, \bar{\alpha} \rangle \in A$. Since f is relevant for $\sigma \preceq \tau$ we have $\sigma' \xrightarrow{\alpha}$. We conclude $(\sigma'(\alpha), \tau'(\alpha)) \in \mathscr{W}$ by definition of \mathscr{W} .

Now we prove that \preceq is the largest coinductive weak subcontract, by showing that every coinductive weak subcontract is included in it. Let \mathscr{W} be a coinductive weak subcontract relation such that $(\sigma, \tau) \in \mathscr{W}$ and assume $\rho \dashv\vdash \sigma$. Let $A(\sigma', \tau')$ stand for the set A of orchestration actions satisfying conditions (1) and (2) of Definition 15 whenever $(\sigma', \tau') \in \mathscr{W}$. Let

$$f(\sigma', \tau') \stackrel{\text{def}}{=} \bigvee_{\langle \alpha, \bar{\alpha} \rangle \in A(\sigma', \tau')} \langle \alpha, \bar{\alpha} \rangle . f(\sigma'(\alpha), \tau'(\alpha)) \quad (4)$$

and let $f \stackrel{\text{def}}{=} f(\sigma, \tau)$. We prove $f : \rho \dashv\vdash \tau$. Consider a derivation $\rho \parallel_f \tau \Longrightarrow \rho' \parallel_{f'} \tau' \dashrightarrow$. By “unzipping” this derivation we obtain that there exist $\alpha_1, \dots, \alpha_n$ such that $\rho \xrightarrow{\bar{\alpha}_1 \dots \bar{\alpha}_n} \rho' \dashrightarrow$ and $f \xrightarrow{\langle \alpha_1, \bar{\alpha}_1 \rangle \dots \langle \alpha_n, \bar{\alpha}_n \rangle} f'$ and $\tau \xrightarrow{\alpha_1 \dots \alpha_n} \tau' \dashrightarrow$. By condition (2) of Definition 15 and by induction on n we derive that $\sigma \xrightarrow{\alpha_1 \dots \alpha_n}$ and $(\sigma(\alpha_1 \dots \alpha_n), \tau(\alpha_1 \dots \alpha_n)) \in \mathscr{W}$. Observe that $\tau(\alpha_1 \dots \alpha_n) \Downarrow \text{init}(\tau')$. By condition (1) of Definition 15 we have that there exists R such that $\sigma(\alpha_1 \dots \alpha_n) \Downarrow R$ and $R \subseteq \text{init}(f') \circ \text{init}(\tau')$, hence there exists σ' such that $\sigma \xrightarrow{\alpha_1 \dots \alpha_n} \sigma' \dashrightarrow$ and $\text{init}(\sigma') \subseteq \text{init}(f') \circ \text{init}(\tau')$. By “zipping” the derivations starting from ρ and σ we obtain $\rho \parallel \sigma \Longrightarrow \rho' \parallel \sigma'$. Furthermore $\rho' \parallel \sigma'$ because $\text{init}(\sigma') \subseteq \text{init}(f') \circ \text{init}(\tau')$. From $\rho \dashv\vdash \sigma$ we conclude $\rho' \xrightarrow{e}$. \square

Proof (of Theorem 5). Let $\mathscr{W} \stackrel{\text{def}}{=} \{(\sigma, \tau) \mid \exists f : f : \sigma \leq \tau\}$. We prove that \mathscr{W} is a coinductive weak subcontract relation by showing that $f : \sigma \leq \tau$ implies

1. $\tau \Downarrow S$ implies $\sigma \Downarrow R$ and $R \subseteq \text{init}(f) \circ S$, and
2. $\tau \xrightarrow{\alpha}$ and $\langle \alpha, \bar{\alpha} \rangle \in \text{init}(f)$ implies $\sigma \xrightarrow{\alpha}$ and $f' : \sigma(\alpha) \leq \tau(\alpha)$ where $f \xrightarrow{\langle \alpha, \bar{\alpha} \rangle} f'$.

We proceed by induction on the maximum depth of an axiom or of an unnested instance of rule (PREFIX) in the derivation tree of $f : \sigma \leq \tau$ and by cases on the last rule applied.

Assume the last rule was (PREFIX). Then $f = \langle \alpha, \bar{\alpha} \rangle . f'$, $\sigma \equiv \alpha . \sigma'$, $\tau \equiv \alpha . \tau'$, and $f' : \sigma' \leq \tau'$ is derivable (we use \equiv to denote syntactic equality up to associativity and commutativity of \oplus). Suppose $\tau \Downarrow S$. Then $S = \{\alpha\}$ and we notice that $\sigma \Downarrow \{\alpha\}$ and $\text{init}(f) \circ S = \{\alpha\}$. We also notice that $\tau \xrightarrow{\alpha}$ and $\langle \alpha, \bar{\alpha} \rangle \in \text{init}(f)$ and $\sigma \xrightarrow{\alpha}$ and that this is the only possible transition for σ and τ . We conclude by observing that $\sigma(\alpha) \equiv \sigma'$ and $\tau(\alpha) \equiv \tau'$ and that $f' : \sigma' \leq \tau'$ is derivable by hypothesis.

Assume the last rule was (RED). Then $\sigma \equiv \sigma' \oplus \tau$ and $f = I(\tau)$. Suppose $\tau \Downarrow S$. Then $\text{init}(f) \circ S = S$ and $\sigma \Downarrow S$. Suppose $\tau \xrightarrow{\alpha}$ and $\langle \alpha, \bar{\alpha} \rangle \in \text{init}(f)$. We distinguish two subcases: if $\sigma' \xrightarrow{\alpha}$, then $\sigma(\alpha) \equiv \sigma'(\alpha) \oplus \tau(\alpha)$ and we conclude $f' : \sigma'(\alpha) \oplus \tau(\alpha) \leq \tau(\alpha)$ by (RED) since $f' = I(\tau(\alpha))$. If $\sigma' \not\xrightarrow{\alpha}$, then $\sigma(\alpha) \equiv \tau(\alpha)$, hence we conclude by reflexivity of \leq (indeed $\sigma = \sigma \oplus \sigma$ and $I(\sigma) : \sigma \oplus \sigma \leq \sigma$).

Assume the last rule was (DEPTH). Then $\sigma \equiv 0$ and $f = 0$. The condition on ready sets of τ trivially holds because $\sigma \Downarrow \emptyset$ and there is nothing left to prove since $\text{init}(f) = \emptyset$.

Assume the last rule was (WEAK). Then $f = f_1 \vee f_2$, $f_1 : \sigma \leq \tau$, and $f_2 \wedge I(\tau) \leq f_1$. Suppose $\tau \Downarrow S$. By induction hypothesis we have $\sigma \Downarrow R$ where $R \subseteq \text{init}(f_1) \circ S$. We conclude $R \subseteq \text{init}(f) \circ S$ by observing that $\text{init}(f_1) \subseteq \text{init}(f)$. Suppose $\tau \xrightarrow{\alpha}$ and $\langle \alpha, \bar{\alpha} \rangle \in \text{init}(f)$. Then $\langle \alpha, \bar{\alpha} \rangle \in \text{init}(f \wedge I(\tau)) = \text{init}((f_1 \vee f_2) \wedge I(\tau)) = \text{init}((f_1 \wedge I(\tau)) \vee (f_2 \wedge I(\tau))) \subseteq \text{init}(f_1)$ since $f_2 \wedge I(\tau) \leq f_1$. By induction hypothesis we have $\sigma \xrightarrow{\alpha}$ and $f'_1 : \sigma(\alpha) \leq \tau(\alpha)$ where $f_1 \xrightarrow{\langle \alpha, \bar{\alpha} \rangle} f'_1$. We distinguish two subcases. If $f_2 \xrightarrow{\langle \alpha, \bar{\alpha} \rangle}$, then $f' = f'_1 \vee f'_2$ where $f_2 \xrightarrow{\langle \alpha, \bar{\alpha} \rangle} f'_2$. From $f_2 \wedge I(\tau) \leq f_1$ we deduce $f'_2 \wedge I(\tau(\alpha)) \leq f'_1$, hence we conclude $f'_1 \vee f'_2 : \sigma(\alpha) \leq \tau(\alpha)$ by rule (WEAK). If $f_2 \not\xrightarrow{\langle \alpha, \bar{\alpha} \rangle}$, then $f' = f'_1$ and there is nothing left to prove.

Assume the last rule was (TRANS). Then $f = f_1 \wedge f_2$, $f_1 : \sigma \leq \sigma'$, $f_2 : \sigma' \leq \tau$ for some σ' . Suppose $\tau \Downarrow S$. By induction hypothesis $\sigma' \Downarrow R'$ for some $R' \subseteq \text{init}(f_2) \circ S$. Again by induction hypothesis $\sigma \Downarrow R$ for some $R \subseteq \text{init}(f_1) \circ R' \subseteq \text{init}(f_1) \circ \text{init}(f_2) \circ S = \text{init}(f) \circ S$. Suppose $\tau \xrightarrow{\alpha}$ and $\langle \alpha, \bar{\alpha} \rangle \in \text{init}(f)$. Then $\langle \alpha, \bar{\alpha} \rangle \in \text{init}(f_1) \cap \text{init}(f_2)$. By induction hypothesis $\sigma' \xrightarrow{\alpha}$ and $f'_2 : \sigma'(\alpha) \leq \tau(\alpha)$ where $f_2 \xrightarrow{\langle \alpha, \bar{\alpha} \rangle} f'_2$. Again by induction hypothesis $f'_1 : \sigma(\alpha) \leq \sigma'(\alpha)$. By (TRANS) we have that $f'_1 \wedge f'_2 : \sigma(\alpha) \leq \tau(\alpha)$ is derivable and we conclude by observing that $f' = f'_1 \wedge f'_2$.

Assume the last rule was (INT). Then $\sigma \equiv \sigma_1 \oplus \sigma_2$, $\tau \equiv \tau_1 \oplus \tau_2$, $f : \sigma_1 \leq \tau_1$, and $f : \sigma_2 \leq \tau_2$. Suppose $\tau \Downarrow S$. Then $\tau_i \Downarrow S$ for some $i \in \{1, 2\}$. By induction hypothesis we deduce $\sigma_i \Downarrow R$ for some $R \subseteq \text{init}(f) \circ S$ and we conclude by observing that $\sigma \Downarrow R$. Suppose $\tau \xrightarrow{\alpha}$ and $\langle \alpha, \bar{\alpha} \rangle \in \text{init}(f)$. We distinguish three interesting subcases, depending on which contracts admit α -successors. Assume $\tau_1 \xrightarrow{\alpha}$ and $\tau_2 \not\xrightarrow{\alpha}$. By induction hypothesis we deduce $\sigma_1 \xrightarrow{\alpha}$ and $\sigma_2 \not\xrightarrow{\alpha}$ and $f' : \sigma_i(\alpha) \leq \tau_i(\alpha)$ for every $i \in \{1, 2\}$. By rule (INT) we deduce $f' : \sigma_1(\alpha) \oplus \sigma_2(\alpha) \leq \tau_1(\alpha) \oplus \tau_2(\alpha)$ and we conclude $f' :$

$\sigma(\alpha) \leq \tau(\alpha)$ by observing that $\sigma(\alpha) \equiv \sigma_1(\alpha) \oplus \sigma_2(\alpha)$ and $\tau(\alpha) \equiv \tau_1(\alpha) \oplus \tau_2(\alpha)$. Assume $\tau_1 \xrightarrow{\alpha}$ and $\sigma_2 \not\xrightarrow{\alpha}$. Then $\tau_2 \not\xrightarrow{\alpha}$. By induction hypothesis we deduce $\sigma_1 \xrightarrow{\alpha}$ and $f' : \sigma_1(\alpha) \leq \tau_1(\alpha)$. We conclude by observing that $\sigma(\alpha) \equiv \sigma_1(\alpha)$ and $\tau(\alpha) \equiv \tau_1(\alpha)$. Assume $\tau_1 \xrightarrow{\alpha}$ and $\tau_2 \xrightarrow{\alpha}$ and $\sigma_2 \xrightarrow{\alpha}$. By induction hypothesis we deduce $\sigma_1 \xrightarrow{\alpha}$ and $f' : \sigma_1(\alpha) \leq \tau_1(\alpha)$. By rule (RED) we have $I(\sigma_1(\alpha)) : \sigma_1(\alpha) \oplus \sigma_2(\alpha) \leq \sigma_1(\alpha)$. From $f' \wedge I(\sigma_1(\alpha)) \leq I(\sigma_1(\alpha))$ and by rule (WEAK) we obtain $I(\sigma_1(\alpha)) \vee f' : \sigma_1(\alpha) \oplus \sigma_2(\alpha) \leq \sigma_1(\alpha)$. By rule (TRANS) we deduce $f' : \sigma_1(\alpha) \oplus \sigma_2(\alpha) \leq \tau_1(\alpha)$. We conclude by observing that $\sigma(\alpha) \equiv \sigma_1(\alpha) \oplus \sigma_2(\alpha)$ and $\tau(\alpha) \equiv \tau_1(\alpha) \oplus \tau_2(\alpha)$.

Assume the last rule was (EXT). Then we can proceed as for the previous case, the only thing that changes being the reasoning on ready sets. The details are left to the reader. \square

Proof (of Lemma 1). In the rewritings that follow we indicate only the most relevant laws that are applied. As regards (S1):

$$\begin{aligned} \sigma \oplus \tau &= (\sigma \oplus \tau) + (\sigma \oplus \tau) \quad (1) \\ &= \sigma \oplus \tau \oplus (\sigma + \tau) \quad (2) \end{aligned}$$

where (1) is justified by (E1) and (2) is justified by (D1).

As regards (S2):

$$\begin{aligned} \sigma \oplus (\sigma + \tau + \rho) &= \sigma + (\sigma \oplus \tau) + (\sigma \oplus \rho) \quad (1) \\ &= \sigma + (\sigma \oplus (\sigma + \tau) \oplus (\sigma + \rho) \oplus (\tau + \rho)) \quad (2) \\ &= \sigma \oplus (\sigma + \tau) \oplus (\sigma + \rho) \oplus (\sigma + \tau + \rho) \quad (3) \\ &= \sigma \oplus (\sigma + \tau) \oplus (\sigma + \tau) \oplus (\sigma + \rho) \oplus (\sigma + \tau + \rho) \quad (4) \\ &= \sigma \oplus (\sigma + \tau) \oplus (\sigma + \tau + \rho) \quad (5) \end{aligned}$$

where (1) is justified by (D2), (2) is justified by (D1), (3) is justified by (D2), (4) is justified by (I1) and finally (5) is justified by rewriting the subterm of step (3) with the original one.

As regards (CO):

$$\begin{aligned} &(\alpha.\sigma + \tau) \oplus (\alpha.\sigma' + \tau') \\ &= (\alpha.\sigma + \tau) \oplus (\alpha.\sigma' + \tau') \oplus (\alpha.\sigma + \alpha.\sigma' + \tau + \tau') \quad (1) \\ &= (\alpha.\sigma + \tau) \oplus (\alpha.\sigma' + \tau') \oplus (\alpha.\sigma + \alpha.\sigma' + \tau + \tau') \\ &\quad \oplus (\alpha.\sigma + \alpha.\sigma' + \tau) \oplus (\alpha.\sigma + \alpha.\sigma' + \tau') \quad (2) \\ &= (\alpha.\sigma + \tau) \oplus (\alpha.\sigma' + \tau') \oplus (\alpha.\sigma + \alpha.\sigma' + \tau + \tau') \\ &\quad \oplus (\alpha.(\sigma \oplus \sigma') + \tau) \oplus (\alpha.(\sigma \oplus \sigma') + \tau') \quad (3) \\ &= (\alpha.\sigma + \tau) \oplus (\alpha.\sigma' + \tau') \oplus (\alpha.(\sigma \oplus \sigma') + \tau) \oplus (\alpha.(\sigma \oplus \sigma') + \tau') \quad (4) \\ &= ((\alpha.\sigma \oplus \alpha.(\sigma \oplus \sigma')) + \tau) \oplus ((\alpha.\sigma' \oplus \alpha.(\sigma \oplus \sigma')) + \tau') \quad (5) \\ &= (\alpha.(\sigma \oplus \sigma') + \tau) \oplus (\alpha.(\sigma \oplus \sigma') + \tau') \quad (6) \end{aligned}$$

where (1) is justified by (S1), (2) is justified by (S2), (3) is justified by (D3), (4) is justified by (S1), (5) is justified by (D1), and (6) is justified by (D4) and (I1).

Proving (E-PREFIX) is trivial. As regards (E-EXT), observe that from $I(\sigma') : \sigma \leq \sigma'$ and $I(\tau') \wedge I(\sigma') \leq I(\sigma')$ we derive $I(\sigma') \vee I(\tau') : \sigma \leq \sigma'$ by an application of (WEAK). Similarly we can derive $I(\sigma') \vee I(\tau') : \tau \leq \tau'$, hence we can apply (EXT) and derive

$I(\sigma') \vee I(\tau') : \sigma + \tau \leq \sigma' + \tau'$. By a similar argument we can also derive $I(\sigma) \vee I(\tau) : \sigma' + \tau' \leq \sigma + \tau$, hence $\sigma + \tau = \sigma' + \tau'$. Rule (E-INT) is analogous.

As regards (WIDTH), from the axiom $0 : 0 \leq \tau$ and the hypothesis $I(\sigma) \wedge I(\tau) \leq 0$ we derive $I(\sigma) : 0 \leq \tau$. From $I(\sigma) : \sigma \leq \sigma$ and applying (EXT) we conclude $I(\sigma) : \sigma + 0 \leq \sigma + \tau$, hence $I(\sigma) : \sigma \leq \sigma + \tau$. \square

Proof (of Lemma 2). We define the *head normal form* of σ as

$$\text{hnf}(\sigma) \stackrel{\text{def}}{=} \bigoplus_{R \in \mathcal{R}(\sigma)} \sum_{\alpha \in R} \alpha \cdot \sigma(\alpha)$$

It is sufficient to prove that $\sigma = \text{hnf}(\sigma)$ is derivable. The statement of the Lemma then follows by a simple induction on the maximum number of nested prefixes in σ . We prove $\sigma = \text{hnf}(\sigma)$ by induction on the maximum depth of a topmost prefix in σ and by cases on the structure of σ . If $\sigma \equiv 0$, then σ is already in head normal form.

If $\sigma \equiv \alpha \cdot \sigma'$, then σ is already in head normal form because $\sigma(\alpha)$ is σ' .

If $\sigma \equiv \sigma_1 + \sigma_2$, then

$$\begin{aligned} \sigma &= \left(\bigoplus_{R_1 \in \mathcal{R}(\sigma_1)} \sum_{\alpha \in R_1} \alpha \cdot \sigma_1(\alpha) \right) + \left(\bigoplus_{R_2 \in \mathcal{R}(\sigma_2)} \sum_{\beta \in R_2} \beta \cdot \sigma_2(\beta) \right) & (1) \\ &= \bigoplus_{R_1 \in \mathcal{R}(\sigma_1), R_2 \in \mathcal{R}(\sigma_2)} \left(\sum_{\alpha \in R_1} \alpha \cdot \sigma_1(\alpha) + \sum_{\beta \in R_2} \beta \cdot \sigma_2(\beta) \right) & (2) \\ &= \bigoplus_{R_1 \in \mathcal{R}(\sigma_1), R_2 \in \mathcal{R}(\sigma_2)} \sum_{\alpha \in R_1 \cup R_2} \alpha \cdot \sigma(\alpha) & (3) \\ &= \bigoplus_{R \in \mathcal{R}(\sigma)} \sum_{\alpha \in R} \alpha \cdot \sigma(\alpha) & (4) \end{aligned}$$

where (1) is justified by the induction hypothesis and congruence rules, (2) is justified by the repeated use of (D1), (3) is justified by (CO), and (4) follows from $\mathcal{R}(\sigma) = \{R_1 \cup R_2 \mid R_1 \in \mathcal{R}(\sigma_1), R_2 \in \mathcal{R}(\sigma_2)\}$. Indeed, if $R \in \mathcal{R}(\sigma)$, then there exist R'_1 and R'_2 such that $\sigma_1 \Downarrow R'_1$ and $\sigma_2 \Downarrow R'_2$ and $R'_1 \cup R'_2 \subseteq R$. Now $R'_1 \subseteq R \cap \text{init}(\sigma_1) \subseteq \text{init}(\sigma_1)$ and $R'_2 \subseteq R \cap \text{init}(\sigma_2) \subseteq \text{init}(\sigma_2)$, hence $R \cap \text{init}(\sigma_1) \in \mathcal{R}(\sigma_1)$ and $R \cap \text{init}(\sigma_2) \in \mathcal{R}(\sigma_2)$. We conclude by observing that $(R \cap \text{init}(\sigma_1)) \cup (R \cap \text{init}(\sigma_2)) = R$ because $R \subseteq \text{init}(\sigma_1) \cup \text{init}(\sigma_2)$. On the other hand, let $R_1 \in \mathcal{R}(\sigma_1)$ and $R_2 \in \mathcal{R}(\sigma_2)$. Then there exist ready sets R'_1 and R'_2 of respectively σ_1 and σ_2 such that $R'_1 \subseteq R_1 \subseteq \text{init}(\sigma_1)$ and $R'_2 \subseteq R_2 \subseteq \text{init}(\sigma_2)$. Hence $R'_1 \cup R'_2 \subseteq R_1 \cup R_2 \subseteq \text{init}(\sigma_1) \cup \text{init}(\sigma_2)$ and we conclude $R_1 \cup R_2 \in \mathcal{R}(\sigma)$ by observing that $\sigma \Downarrow R'_1 \cup R'_2$ and $\text{init}(\sigma) = \text{init}(\sigma_1) \cup \text{init}(\sigma_2)$.

Finally, if $\sigma \equiv \sigma_1 \oplus \sigma_2$, then

$$\begin{aligned} \sigma &= \left(\bigoplus_{R_1 \in \mathcal{R}(\sigma_1)} \sum_{\alpha \in R_1} \alpha \cdot \sigma_1(\alpha) \right) \oplus \left(\bigoplus_{R_2 \in \mathcal{R}(\sigma_2)} \sum_{\beta \in R_2} \beta \cdot \sigma_2(\beta) \right) & (1) \\ &= \left(\bigoplus_{R_1 \in \mathcal{R}(\sigma_1)} \sum_{\alpha \in R_1} \alpha \cdot \sigma(\alpha) \right) \oplus \left(\bigoplus_{R_2 \in \mathcal{R}(\sigma_2)} \sum_{\beta \in R_2} \beta \cdot \sigma(\beta) \right) & (2) \\ &= \bigoplus_{R \in \mathcal{R}(\sigma)} \sum_{\alpha \in R} \alpha \cdot \sigma(\alpha) & (3) \end{aligned}$$

where (1) is justified by the induction hypothesis and congruence rules, (2) is justified by (CO), and (3) is justified by the repeated use of (S1) and (S2). \square

Proof (of Theorem 6). By Lemma 2 we can assume that σ and τ are in normal form. We reason by induction on the depth of σ and τ .

If $\tau \equiv 0$, then σ must have an empty ready set hence by (MUST) we have $0 : \sigma \leq 0$ and we conclude $f : \sigma \leq \tau$ by (WEAK) because $f \wedge I(0) \leq 0$.

For the remaining cases, assume

$$\sigma \equiv \bigoplus_{R \in \mathcal{R}(\sigma)} \sum_{\alpha \in R} \alpha \cdot \sigma_\alpha \quad \text{and} \quad \tau \equiv \bigoplus_{S \in \mathcal{R}(\tau)} \sum_{\alpha \in S} \alpha \cdot \tau_\alpha$$

and assume $\tau \xrightarrow{\alpha}$ and $f \xrightarrow{\langle \alpha, \bar{\alpha} \rangle}$. From $f : \sigma \preceq \tau$ we deduce $\sigma \xrightarrow{\alpha}$ and by induction hypothesis we derive

$$f' : \sigma_\alpha \leq \tau_\alpha$$

where $f \xrightarrow{\langle \alpha, \bar{\alpha} \rangle} f'$. Then, by (PREFIX),

$$\langle \alpha, \bar{\alpha} \rangle . f' : \alpha . \sigma_\alpha \leq \alpha . \tau_\alpha .$$

Now assume $\tau \Downarrow R$. From $f : \sigma \preceq \tau$ and the fact that σ and τ are in head normal form we have $\sigma \Downarrow \text{init}(f) \circ R$. Let $f_R \stackrel{\text{def}}{=} \bigvee_{f \xrightarrow{\langle \alpha, \bar{\alpha} \rangle} f'', \alpha \in R} \langle \alpha, \bar{\alpha} \rangle . f''$ and notice that $f_R \wedge \langle \alpha, \bar{\alpha} \rangle . I(\tau_\alpha) \leq \langle \alpha, \bar{\alpha} \rangle . f'$. Hence, by (WEAK),

$$f_R : \alpha . \sigma_\alpha \leq \alpha . \tau_\alpha$$

and, by (EXT),

$$f_R : \sum_{\alpha \in R, f \xrightarrow{\langle \alpha, \bar{\alpha} \rangle}} \alpha . \sigma_\alpha \leq \sum_{\alpha \in R, f \xrightarrow{\langle \alpha, \bar{\alpha} \rangle}} \alpha . \tau_\alpha .$$

From $\text{init}(f) \circ R \subseteq R$ and by applying (WIDTH),

$$f_R : \sum_{\alpha \in R, f \xrightarrow{\langle \alpha, \bar{\alpha} \rangle}} \alpha . \sigma_\alpha \leq \sum_{\alpha \in R} \alpha . \tau_\alpha .$$

Let $f' \stackrel{\text{def}}{=} \bigvee_{\tau \Downarrow R'} f'_R$. From $(\bigcup_{\tau \Downarrow R'} \text{init}(f) \circ R') \cap R = (\text{init}(f) \circ \bigcup_{\tau \Downarrow R'} R') \cap R \subseteq \text{init}(f) \circ R$ we observe that $f' \wedge \bigvee_{\alpha \in R} \langle \alpha, \bar{\alpha} \rangle . I(\tau_\alpha) \leq f_R$. Hence, by (WEAK), by iterating over all the ready sets of τ , and by (INT), we obtain

$$f' : \bigoplus_{\tau \Downarrow R} \sum_{\alpha \in R, f \xrightarrow{\langle \alpha, \bar{\alpha} \rangle}} \alpha . \sigma_\alpha \leq \tau .$$

Now

$$f' : \sigma \leq \bigoplus_{\tau \Downarrow R} \sum_{\alpha \in R, f \xrightarrow{\alpha}} \alpha . \sigma_\alpha$$

by possibly applying (RED) for removing all the ready sets of σ that are not in $\{\text{init}(f) \circ S \mid \tau \Downarrow S\}$ hence, by (TRANS), we conclude $f' : \sigma \leq \tau$. In order to prove $f : \sigma \leq \tau$ it is sufficient to apply (WEAK). This is possible because $f \wedge I(\tau) \leq f'$. Indeed, assume $\tau \xrightarrow{\alpha}$ and $\langle \alpha, \bar{\alpha} \rangle \in \text{init}(f)$. Then $\alpha \in R$ for some $\tau \Downarrow R$, hence $\sigma \Downarrow \text{init}(f) \circ R$ and now $\alpha \in \text{init}(f) \circ R$. So, it must be $f_R \xrightarrow{\langle \alpha, \bar{\alpha} \rangle}$ from which we conclude $f' \xrightarrow{\langle \alpha, \bar{\alpha} \rangle}$. \square

A.3 Proofs of §4

Proof (of Proposition 7). We say that a subterm $\bar{\alpha} . \rho'$ of ρ is useless if $\rho \xrightarrow{\bar{\alpha}} \bar{\alpha} \rightarrow \rho'$ and $\sigma \xrightarrow{\bar{\alpha}}$ implies $\sigma(\bar{\alpha}) \not\xrightarrow{\bar{\alpha}}$. Let ρ_r be the (client) contract obtained from ρ by replacing every useless subterm $\bar{\alpha} . \rho'$ with $\bar{\alpha} . 0$. Clearly $\rho_r \dashv \sigma$ since no synchronization will ever occur on those $\bar{\alpha}$ actions that guard useless subterms of ρ . From the hypothesis $\sigma \preceq_k \tau$ there exists a k -orchestrator g such that $g : \rho_r \dashv \tau$. Let

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

and let $g_r \stackrel{\text{def}}{=} R(g, \sigma, \tau)$. Observe that g_r is relevant for $\sigma \preceq_k \tau$ by its own definition and $g_r : \rho_r \dashv\!\! \dashv \tau$ because every derivation starting from $\rho_r \parallel_{g_r} \tau$ is also a possible derivation starting from $\rho_r \parallel_g \tau$. We prove that $g_r : \rho \dashv\!\! \dashv \tau$. Consider a derivation $\rho \parallel_{g_r} \tau \Longrightarrow \rho' \parallel_{g'_r} \tau' \dashv\!\! \dashv$. Then there exist $\varphi_1, \dots, \varphi_n$ and $\varphi'_1, \dots, \varphi'_n$ such that $\rho \xrightarrow{\overline{\varphi_1 \dots \varphi_n}} \rho' \dashv\!\! \dashv$ and $g_r \xrightarrow{\langle \varphi_1, \overline{\varphi'_1} \rangle \dots \langle \varphi_n, \overline{\varphi'_n} \rangle} g'_r$ and $\tau \xrightarrow{\varphi'_1 \dots \varphi'_n} \tau' \dashv\!\! \dashv$. 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 $\rho_r \xrightarrow{\overline{\varphi_1 \dots \varphi_n}} \rho'_r$ and $\text{init}(\rho'_r) = \text{init}(\rho')$ (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 $\overline{\alpha}.0$). By zipping the derivations starting from ρ_r, g_r , and τ we obtain $\rho_r \parallel_{g_r} \tau \Longrightarrow \rho'_r \parallel_{g'_r} \tau'$ and we notice that $\rho'_r \parallel_{g'_r} \tau' \dashv\!\! \dashv$ since $\text{init}(\rho'_r) = \text{init}(\rho')$. From $g_r : \rho_r \dashv\!\! \dashv \tau$ we deduce $\rho'_r \xrightarrow{e}$, hence we conclude $\rho' \xrightarrow{e}$. \square

Proof (of Proposition 8). The “if” part is trivial. As regards the “only if” part, we use the same intuition as for the synchronous case, by considering the most demanding client σ^\top that is satisfied by σ . Let f be a k -orchestrator such that $f : \sigma^\top \dashv\!\! \dashv \tau$ 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 $\rho \dashv\!\! \dashv \sigma$ and $f : \rho \not\vdash \tau$. Then there exists a derivation $\rho \parallel_f \tau \Longrightarrow \rho' \parallel_{f'} \tau' \dashv\!\! \dashv$ such that $\rho' \dashv\!\! \dashv$. Consequently there exist $\varphi_1, \dots, \varphi_n$ and $\varphi'_1, \dots, \varphi'_n$ such that $\rho \xrightarrow{\overline{\varphi_1 \dots \varphi_n}} \rho' \dashv\!\! \dashv$ and $\tau \xrightarrow{\varphi'_1 \dots \varphi'_n} \tau' \dashv\!\! \dashv$ and $f \xrightarrow{\langle \varphi_1, \overline{\varphi'_1} \rangle \dots \langle \varphi_n, \overline{\varphi'_n} \rangle} f'$. Since f is relevant, it is easy to deduce, by induction on n , that $\sigma \xrightarrow{\varphi_1 \dots \varphi_n}$. From $\rho' \parallel_{f'} \tau' \dashv\!\! \dashv$ we deduce that $\rho' \dashv\!\! \dashv$ implies $f' \xrightarrow{\langle \alpha, \varepsilon \rangle}$ and $f' \xrightarrow{\langle \alpha, \overline{\alpha} \rangle}$ implies $\tau' \dashv\!\! \dashv$. Let R_1, \dots, R_m be the ready sets of $\sigma(\varphi_1 \dots \varphi_n)$. From $\rho \dashv\!\! \dashv \sigma$ and $\rho' \dashv\!\! \dashv$ we deduce that for every $1 \leq i \leq n$ there exists $\alpha_i \in R_i$ and $\rho' \dashv\!\! \dashv$. By definition of most demanding client we have $\sigma(\varphi_1 \dots \varphi_n)^\top \Downarrow \{\overline{\alpha}_1, \dots, \overline{\alpha}_m\}$, because each ready set of $\sigma(\varphi_1 \dots \varphi_n)^\top$ is obtained by taking one action from every non-empty ready set of $\sigma(\varphi_1 \dots \varphi_n)$. Hence there exists ρ'' such that $\sigma^\top \xrightarrow{\overline{\varphi_1 \dots \varphi_n}} \rho'' \dashv\!\! \dashv$ and $\text{init}(\rho'') = \{\overline{\alpha}_1, \dots, \overline{\alpha}_m\} \subseteq \text{init}(\rho')$. By zipping the derivations starting from σ^\top, f , and τ we obtain a derivation $\sigma^\top \parallel_f \tau \Longrightarrow \rho'' \parallel_{f'} \tau' \dashv\!\! \dashv$ where $\rho'' \dashv\!\! \dashv$. This is absurd, for $f : \sigma^\top \dashv\!\! \dashv \tau$ by hypothesis. \square

Proof (of Theorem 7). (\Rightarrow) Assume $f : \rho \dashv\!\! \dashv_k \sigma$ and consider a derivation $\rho \parallel_f(\sigma) \Longrightarrow \rho' \parallel \tau \dashv\!\! \dashv$. Then there exist φ, φ' , and f' such that $\rho \xrightarrow{\overline{\varphi}} \rho' \dashv\!\! \dashv$ and $f(\sigma) \xrightarrow{\varphi} f'(\sigma(\varphi')) \Longrightarrow \tau \dashv\!\! \dashv$. By definition of $f(\sigma)$ there exist $\varphi_1, \dots, \varphi_n$ and $\varphi'_1, \dots, \varphi'_n$ such that $\varphi = \varphi_1 \dots \varphi_n$ and $\varphi' = \varphi'_1 \dots \varphi'_n$ and $f \xrightarrow{\langle \varphi_1, \overline{\varphi'_1} \rangle \dots \langle \varphi_n, \overline{\varphi'_n} \rangle} f'$. From $\rho' \parallel \tau \dashv\!\! \dashv$ we deduce $\overline{\text{init}(\rho')} \cap \text{init}(\tau) = \emptyset$. From $f'(\sigma(\varphi')) \Longrightarrow \tau \dashv\!\! \dashv$ and by definition of $f'(\sigma(\varphi'))$ we deduce that there exist $\alpha_1, \dots, \alpha_m$ and σ' and f'' such that $\sigma \xrightarrow{\varphi'_1 \alpha_1 \dots \alpha_m} \sigma' \dashv\!\! \dashv$ and $f' \xrightarrow{\langle \varepsilon, \overline{\alpha}_1 \rangle \dots \langle \varepsilon, \overline{\alpha}_m \rangle} f''$ and $(\emptyset \bullet \text{init}(f'')) \cap \text{init}(\sigma') = \emptyset$ and $\text{init}(f'') \bullet \text{init}(\sigma') \subseteq \text{init}(\tau)$. By zipping the derivations starting from ρ, f , and σ we obtain $\rho \parallel_f \sigma \Longrightarrow \rho' \parallel_{f''} \sigma'$. Furthermore $\rho' \parallel_{f''} \sigma' \dashv\!\! \dashv$ because $\rho' \dashv\!\! \dashv$ and $\sigma' \dashv\!\! \dashv$ and $\overline{\text{init}(\rho')} \cap (\text{init}(f'') \bullet \text{init}(\sigma')) \subseteq \overline{\text{init}(\rho')} \cap \text{init}(\tau) = \emptyset$. From $f : \rho \dashv\!\! \dashv_k \sigma$ we conclude $\rho' \xrightarrow{e}$.

(\Leftarrow) Assume $\rho \dashv f(\sigma)$ and consider a derivation $\rho \parallel_f \sigma \Longrightarrow \rho' \parallel_{f'} \sigma' \dashv$. By “un-zipping” this derivation we have that there exist $\varphi_1, \dots, \varphi_n$ and $\varphi'_1, \dots, \varphi'_n$ such that $\rho \xrightarrow{\langle \overline{\varphi}_1, \dots, \overline{\varphi}_n \rangle} \rho' \dashv$ and $f \xrightarrow{\langle \varphi_1, \overline{\varphi}'_1 \rangle \dots \langle \varphi_n, \overline{\varphi}'_n \rangle} f' \dashv$ and $\sigma \xrightarrow{\langle \varphi'_1, \dots, \varphi'_n \rangle} \sigma' \dashv$. Furthermore from $\rho' \parallel_{f'} \sigma' \dashv$ we derive $\overline{\text{init}(\rho')} \cap (\text{init}(f') \bullet \text{init}(\sigma')) = \emptyset$. By definition of $f(\sigma)$ there exists τ such that $f(\sigma) \xrightarrow{\langle \varphi_1, \dots, \varphi_n \rangle} \tau \dashv$ and $\text{init}(\tau) = \text{init}(f') \bullet \text{init}(\sigma')$. By zipping the derivations starting from ρ and $f(\sigma)$ we obtain $\rho \parallel f(\sigma) \Longrightarrow \rho' \parallel \tau$. Furthermore $\rho' \parallel \tau \dashv$ because $\rho' \dashv$, $\tau \dashv$, and $\overline{\text{init}(\rho')} \cap \text{init}(\tau) = \overline{\text{init}(\rho')} \cap (\text{init}(f') \bullet \text{init}(\sigma')) = \emptyset$. From $\rho \dashv f(\sigma)$ we conclude $\rho' \xrightarrow{e}$. \square

Proof (of Proposition 9). We prove item (1); items (2) and (3) are similar. By Theorem 7 it is sufficient to prove that $f : \rho \dashv \sigma$ implies $f : \rho \dashv \tau$ for every ρ , under the hypothesis $\sigma \sqsubseteq \tau$. Consider a derivation $\rho \parallel_f \tau \Longrightarrow \rho' \parallel_{f'} \tau' \dashv$. Then there exist $\varphi_1, \dots, \varphi_n$ and $\varphi'_1, \dots, \varphi'_n$ such that $\rho \xrightarrow{\langle \overline{\varphi}_1, \dots, \overline{\varphi}_n \rangle} \rho' \dashv$ and $f \xrightarrow{\langle \varphi_1, \overline{\varphi}'_1 \rangle \dots \langle \varphi_n, \overline{\varphi}'_n \rangle} f' \dashv$ and $\tau \xrightarrow{\langle \varphi'_1, \dots, \varphi'_n \rangle} \tau' \dashv$. Furthermore, from $\rho' \parallel_{f'} \tau' \dashv$ we deduce $\overline{\text{init}(\rho')} \cap (\text{init}(f') \bullet \text{init}(\tau')) = \emptyset$. From $\sigma \sqsubseteq \tau$ we derive that there exists σ' such that $\sigma \xrightarrow{\langle \varphi'_1, \dots, \varphi'_n \rangle} \sigma' \dashv$ and $\text{init}(\sigma') \subseteq \text{init}(\tau')$. By zipping the derivations starting from ρ , f , and σ we obtain $\rho \parallel_f \sigma \Longrightarrow \rho' \parallel_{f'} \sigma'$. Furthermore, $\overline{\text{init}(\rho')} \cap (\text{init}(f') \bullet \text{init}(\sigma')) \subseteq \overline{\text{init}(\rho')} \cap (\text{init}(f') \bullet \text{init}(\tau')) = \emptyset$, hence $\rho' \parallel_{f'} \sigma' \dashv$. From $f : \rho \dashv \sigma$ we conclude $\rho' \xrightarrow{e}$. \square

The proof that $f \cdot g$ is the orchestrator we are looking for needs the following technical result, which tells us about the “un-zipping” of compound orchestrators.

Lemma 3. $f \cdot g \xrightarrow{\langle \overline{\psi}_1, \overline{\psi}'_1 \rangle \dots \langle \overline{\psi}_m, \overline{\psi}'_m \rangle} h$ implies that there exist $\varphi_1, \dots, \varphi_n$ and $\varphi'_1, \dots, \varphi'_n$ and $\varphi''_1, \dots, \varphi''_n$ such that $f \xrightarrow{\langle \varphi_1, \overline{\varphi}'_1 \rangle \dots \langle \varphi_n, \overline{\varphi}'_n \rangle} f' \dashv$ and $g \xrightarrow{\langle \varphi'_1, \overline{\varphi}''_1 \rangle \dots \langle \varphi_n, \overline{\varphi}''_n \rangle} g' \dashv$ and $\psi_1 \dots \psi_m = \varphi_1 \dots \varphi_n$ and $\psi'_1 \dots \psi'_m = \varphi'_1 \dots \varphi'_n$ and $\text{init}(f' \cdot g') \subseteq \text{init}(h)$.

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

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

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

$$f \xrightarrow{\langle \varepsilon, \overline{\varphi}'_1 \rangle \dots \langle \varepsilon, \overline{\varphi}'_{n-1} \rangle \langle \overline{\psi}, \overline{\varphi}'_n \rangle} f' \quad \text{and} \quad g \xrightarrow{\langle \varphi'_1, \varepsilon \rangle \dots \langle \varphi'_{n-1}, \varepsilon \rangle \langle \varphi'_n, \overline{\psi}' \rangle} g'$$

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

Proof (of Theorem 8). Let $\rho \dashv f(g(\sigma))$. By Theorem 7 it is sufficient to show that $f \cdot g : \rho \dashv \sigma$, so consider a derivation $\rho \parallel_{f \cdot g} \sigma \Longrightarrow \rho' \parallel_h \sigma' \dashv \dashv$. By unzipping this derivation we deduce that there exist ψ_1, \dots, ψ_m and ψ'_1, \dots, ψ'_m such that $\rho \xrightarrow{\overline{\psi_1 \dots \psi_m}} \rho' \dashv \dashv$ and $f \cdot g \xrightarrow{\langle \psi_1, \overline{\psi'_1} \rangle \dots \langle \psi_m, \overline{\psi'_m} \rangle} h$ and $\sigma \xrightarrow{\overline{\psi'_1 \dots \psi'_m}} \sigma' \dashv \dashv$. From $\rho' \parallel_h \sigma' \dashv \dashv$ we deduce $\overline{\text{init}(\rho')} \cap (\text{init}(h) \bullet \text{init}(\sigma')) = \emptyset$. By Lemma 3 we derive that there exist $\varphi_1, \dots, \varphi_n, \varphi'_1, \dots, \varphi'_n$ and $\varphi''_1, \dots, \varphi''_n$ such that $f \xrightarrow{\langle \varphi_1, \overline{\varphi'_1} \rangle \dots \langle \varphi_n, \overline{\varphi'_n} \rangle} f'$ and $g \xrightarrow{\langle \varphi'_1, \overline{\varphi''_1} \rangle \dots \langle \varphi_n, \overline{\varphi''_n} \rangle} g'$ and $\psi_1 \dots \psi_m = \varphi_1 \dots \varphi_n$ and $\psi'_1 \dots \psi'_m = \varphi'_1 \dots \varphi'_n$ and $\text{init}(f' \cdot g') \subseteq \text{init}(h)$. Since f and g are valid orchestrators of finite rank, there exist f'', g'' , and φ such that $f' \xrightarrow{\langle \varepsilon, \overline{\varphi} \rangle} f''$ and $g' \xrightarrow{\langle \varphi, \varepsilon \rangle} g''$ and $f'' \xrightarrow{\langle \varepsilon, \overline{\alpha} \rangle}$ implies $g'' \xrightarrow{\langle \alpha, \varepsilon \rangle}$. Namely f'' and g'' are two residual orchestrators that do not “synchronize” with each other. By definition of orchestrator composition, observe that $\text{init}(f'' \cdot g'') \subseteq \text{init}(f' \cdot g')$ because $f'' \cdot g''$ is a summand within $f' \cdot g'$. By definition of orchestrator application we have $g(\sigma) \xrightarrow{\overline{\varphi'_1 \dots \varphi'_n}} g'(\sigma(\varphi'_1 \dots \varphi'_n)) \Longrightarrow g'(\sigma') \xrightarrow{\varphi} g''(\sigma')$. Furthermore $\emptyset \circ \text{init}(g'') \subseteq \emptyset \circ \text{init}(f'' \cdot g'') \subseteq \emptyset \circ \text{init}(f' \cdot g') \subseteq \emptyset \circ \text{init}(h)$ hence $(\emptyset \circ \text{init}(g'')) \cap \overline{\text{init}(\sigma')} = \emptyset$ and $g''(\sigma') \dashv \dashv$ and $\text{init}(g''(\sigma')) = \text{init}(g'') \bullet \text{init}(\sigma')$. By definition of orchestrator application we have $f(g(\sigma)) \xrightarrow{\overline{\varphi_1 \dots \varphi_n}} f'(g(\sigma)(\varphi_1 \dots \varphi_n)) = f'(g'(\sigma(\varphi'_1 \dots \varphi'_n))) \Longrightarrow f'(g'(\sigma')) \Longrightarrow f''(g''(\sigma'))$. Now we want to show that $(\emptyset \circ \text{init}(f'')) \cap \overline{\text{init}(g''(\sigma'))} = \emptyset$. From $\text{init}(g''(\sigma')) = \text{init}(g'') \bullet \text{init}(\sigma')$ we derive that $(\emptyset \circ \text{init}(f'')) \cap \overline{\text{init}(g''(\sigma'))} \neq \emptyset$ if and only if there exists α such that $f'' \xrightarrow{\langle \varepsilon, \overline{\alpha} \rangle}$ and either $g'' \xrightarrow{\langle \alpha, \varepsilon \rangle}$ or $(g'' \xrightarrow{\langle \alpha, \overline{\alpha} \rangle}$ and $\sigma' \xrightarrow{\alpha}$). However, by the way f'' and g'' have been chosen we have that $f'' \xrightarrow{\langle \varepsilon, \overline{\alpha} \rangle}$ implies $g'' \xrightarrow{\langle \alpha, \varepsilon \rangle}$. Furthermore, if $f'' \xrightarrow{\langle \varepsilon, \overline{\alpha} \rangle}$ and $g'' \xrightarrow{\langle \alpha, \overline{\alpha} \rangle}$, then $\langle \varepsilon, \overline{\alpha} \rangle \in \text{init}(f'' \cdot g'') \subseteq \text{init}(h)$. Then $\sigma' \xrightarrow{\alpha}$ because $\rho' \parallel_h \sigma' \dashv \dashv$. Hence $(\emptyset \circ \text{init}(f'')) \cap \overline{\text{init}(g''(\sigma'))} = \emptyset$, so $f''(g''(\sigma')) \dashv \dashv$ and $\text{init}(f''(g''(\sigma'))) = \text{init}(f'') \bullet \text{init}(g''(\sigma')) = \text{init}(f'') \bullet \text{init}(g'') \bullet \text{init}(\sigma') = \text{init}(f'' \cdot g'') \bullet \text{init}(\sigma') \subseteq \text{init}(h) \bullet \text{init}(\sigma')$. By zipping the derivations starting from ρ and $f(g(\sigma))$ we obtain $\rho \parallel f(g(\sigma)) \Longrightarrow \rho' \parallel f''(g''(\sigma')) \dashv \dashv$, hence we conclude $\rho' \xrightarrow{e}$. \square

Proof (of Theorem 9). We prove that \preceq_k is a coinductive weak k -subcontract relation. 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

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

is a coinductive k -subcontract relation. Let $(\mathbb{B}, \sigma', \tau') \in \mathscr{W}_k$. Then there exist $\varphi_1, \dots, \varphi_n$ and $\varphi'_1, \dots, \varphi'_n$ and f' such that $f \xrightarrow{\langle \varphi_1, \overline{\varphi'_1} \rangle \dots \langle \varphi_n, \overline{\varphi'_n} \rangle} f'$, $\mathbb{B} = \tilde{\emptyset} \langle \varphi_1, \overline{\varphi'_1} \rangle \dots \langle \varphi_n, \overline{\varphi'_n} \rangle$, $\sigma' = \sigma(\varphi_1 \dots \varphi_n)$, and $\tau' = \tau(\varphi'_1 \dots \varphi'_n)$. Since f is a k -orchestrator we have that \mathbb{B} is a k -buffer. Let $A \stackrel{\text{def}}{=} \text{init}(f')$. As regards condition (1) of Definition 22, let R_1, \dots, R_m be the ready sets of σ' . Assume by contradiction that there exists S such that $\tau' \Downarrow S$ and

$R_i \not\subseteq A \circ S$ for every $1 \leq i \leq m$ and $(\emptyset \bullet A) \cap \bar{S} = \emptyset$. Then there exists $\alpha_i \in R_i \setminus A \circ S$ for every $1 \leq i \leq m$. Let $\rho \stackrel{\text{def}}{=} \sum_{1 \leq i \leq m} \bar{\alpha}_i.e$. We have $\rho \dashv \sigma'$ but $f' : \rho \not\vdash \tau'$, which is absurd. As regards condition (2) of Definition 22, assume $\tau' \xrightarrow{\varphi'}$ and $\langle \varphi, \bar{\varphi}' \rangle \in A$. Since f is relevant we have $\sigma' \xrightarrow{\varphi}$. We conclude $(\mathbb{B}\langle \varphi, \bar{\varphi}' \rangle, \sigma'(\varphi), \tau'(\varphi')) \in \mathcal{W}_k$ by definition of \mathcal{W}_k .

Now we prove that \preceq_k is the largest coinductive weak k -subcontract relation, by showing as usual that every coinductive weak k -subcontract relation is included in it. Let \mathcal{W}_k be a coinductive weak k -subcontract relation such that $(\tilde{\theta}, \sigma, \tau) \in \mathcal{W}_k$ and assume $\rho \dashv \sigma$. Let $A(\mathbb{B}, \sigma', \tau')$ stand for the set A of orchestration actions satisfying conditions (1) and (2) of Definition 22 whenever $(\mathbb{B}, \sigma', \tau') \in \mathcal{W}_k$. Let

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

and let $f \stackrel{\text{def}}{=} f(\tilde{\theta}, \sigma, \tau)$. Observe that f is well defined by regularity of σ and τ and that it is a k -orchestrator. We prove $f : \rho \dashv \tau$. Consider a derivation $\rho \parallel_f \tau \Longrightarrow \rho' \parallel_{f'} \tau' \dashv$. By “unzipping” this derivation we obtain that there exist $\varphi_1, \dots, \varphi_n$ and $\varphi'_1, \dots, \varphi'_n$ such that $\rho \xrightarrow{\bar{\varphi}_1 \cdots \bar{\varphi}_n} \rho' \dashv$ and $f \xrightarrow{\langle \varphi_1, \bar{\varphi}'_1 \rangle \cdots \langle \varphi_n, \bar{\varphi}'_n \rangle} f'$ and $\tau \xrightarrow{\varphi'_1 \cdots \varphi'_n} \tau' \dashv$. By condition (2) of Definition 22 and by induction on n we derive that $\sigma \xrightarrow{\varphi_1 \cdots \varphi_n}$ and $(\tilde{\theta} \langle \varphi_1, \bar{\varphi}'_1 \rangle \cdots \langle \varphi_n, \bar{\varphi}'_n \rangle, \sigma(\varphi_1 \cdots \varphi_n), \tau(\varphi'_1 \cdots \varphi'_n)) \in \mathcal{W}_k$. Observe that $\tau(\varphi'_1 \cdots \varphi'_n) \Downarrow \text{init}(\tau')$. By condition (1) of Definition 22 we have that either there exists R such that $\sigma(\varphi_1 \cdots \varphi_n) \Downarrow R$ and $R \subseteq \text{init}(f') \circ \text{init}(\tau')$ or $(\emptyset \bullet \text{init}(f')) \cap \text{init}(\tau') \neq \emptyset$. However from $\rho' \parallel_{f'} \tau' \dashv$ we derive $(\emptyset \bullet \text{init}(f')) \cap \text{init}(\tau') = \emptyset$, hence there exists σ' such that $\sigma \xrightarrow{\varphi_1 \cdots \varphi_n} \sigma'$ and $\text{init}(\sigma') \subseteq \text{init}(f') \circ \text{init}(\tau')$. By “zip-ping” the derivations starting from ρ and σ we obtain $\rho \parallel \sigma \Longrightarrow \rho' \parallel \sigma'$. Furthermore $\rho' \parallel \sigma' \dashv$ because $\text{init}(\sigma') \subseteq \text{init}(f') \circ \text{init}(\tau')$. From $\rho \dashv \sigma$ we conclude $\rho' \xrightarrow{e}$. \square

A.4 Proofs of §5

Proof (of Proposition 10). First we prove that $\text{viable}(\cdot)$ is a coinductive viability. Let $\text{viable}(\rho)$ and $\rho \Downarrow R$. Then there exists σ such that $\rho \dashv \sigma$ and ρ' such that $\rho \Longrightarrow \rho' \dashv$ and $R = \text{init}(\rho')$. If $\rho' \xrightarrow{e}$ there is nothing to prove, so assume $\rho' \xrightarrow{e'} \sigma' \dashv$. 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 $\rho \Longrightarrow \rho' \xrightarrow{\bar{\alpha}} \rho''$ and $\sigma \Longrightarrow \sigma' \xrightarrow{\alpha} \sigma''$. It is trivial to see that from $\rho \dashv \sigma$ and $\rho \xrightarrow{\bar{\alpha}}$ and $\sigma \xrightarrow{\alpha}$ we have $\rho(\bar{\alpha}) \dashv \sigma(\alpha)$, hence we conclude $\text{viable}(\rho(\bar{\alpha}))$.

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

$$S(\rho) \stackrel{\text{def}}{=} \sum_{\rho \Downarrow R, \alpha \in R \setminus \{e\}, \rho(\alpha) \in \mathcal{V}} \bar{\alpha}.S(\rho(\alpha))$$

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

Proof (of Theorem 10). As regards item (1), consider a derivation $\rho \parallel \rho^\perp \Longrightarrow \rho' \parallel \sigma \dashv\vdash$ and assume by contradiction that $\rho' \not\rightarrow^e$. By unzipping this derivation we obtain that there exists φ such that $\rho \xrightarrow{\varphi} \rho' \dashv\vdash$ and $\rho^\perp \xrightarrow{\bar{\varphi}} \sigma \dashv\vdash$. In particular, by definition of ρ^\perp we can rewrite this latter derivation as $\rho^\perp \xrightarrow{\bar{\varphi}} \rho(\varphi)^\perp \Longrightarrow \sigma \dashv\vdash$. From $\rho' \parallel \sigma \dashv\vdash$ we deduce $\text{init}(\rho') \cap \text{init}(\sigma) = \emptyset$. Let R_1, \dots, R_n be the ready sets of $\rho(\varphi)$ not containing e (there must be at least one since $\rho' \not\rightarrow^e$). From the fact that ρ is viable and by definition of ρ^\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 e and whose continuation is viable. Hence, $\text{init}(\sigma) = \{\bar{\alpha}_1, \dots, \bar{\alpha}_n\}$ where $\alpha_i \in R_i$ and $\rho(\varphi\alpha_i)$ is viable. From $\rho(\varphi) \Longrightarrow \rho' \dashv\vdash$ we deduce that $\text{init}(\rho') = R_k$ for some $k \in \{1, \dots, n\}$. Now $\rho' \xrightarrow{\alpha_k}$ and $\sigma \xrightarrow{\bar{\alpha}_k}$, which contradicts $\text{init}(\rho') \cap \text{init}(\sigma) = \emptyset$.

As regards item (2), it is sufficient to prove that $\mathscr{W} \stackrel{\text{def}}{=} \{(\tilde{\emptyset}, \rho(\bar{\varphi})^\perp, \sigma(\varphi)) \mid \rho \xrightarrow{\bar{\varphi}}, \sigma \xrightarrow{\varphi}\}$ 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 $\rho' = \rho(\bar{\varphi})^\perp$ and $\sigma' = \sigma(\varphi)$. Consider $A \stackrel{\text{def}}{=} \{\langle \alpha, \bar{\alpha} \mid \rho' \xrightarrow{\bar{\alpha}}\}$ and observe that $\tilde{\emptyset} \vdash_0 A$. As regards condition (1) in Definition 18, let $\{R_1, \dots, R_n\} = \{R \mid \rho \Downarrow R, e \notin R\}$ be the ready sets of $\rho(\bar{\varphi})$ not containing e . From the hypothesis $\rho \dashv\vdash \sigma$ we derive $\rho(\bar{\varphi}) \dashv\vdash \sigma(\varphi)$, hence $R_i \cap S \neq \emptyset$ for every $1 \leq i \leq n$. Namely, for every $1 \leq i \leq n$ there exists $\bar{\alpha}_i \in R_i \cap S$. By definition of dual contract we have $\rho(\bar{\varphi})^\perp \Downarrow \{\bar{\alpha}_1, \dots, \bar{\alpha}_n\}$. We conclude $\{\bar{\alpha}_1, \dots, \bar{\alpha}_n\} \subseteq A \circ S$. As regards condition (2), assume $\sigma(\varphi) \xrightarrow{\alpha}$ and $\langle \alpha, \bar{\alpha} \rangle \in A$. Then $\sigma \xrightarrow{\varphi\alpha}$ and $\rho(\bar{\varphi})^\perp \xrightarrow{\alpha}$ hence $\rho \xrightarrow{\bar{\varphi}\alpha}$. By definition of \mathscr{W} we conclude that $(\tilde{\emptyset}, \rho(\bar{\varphi})^\perp(\alpha), \sigma(\varphi)(\alpha)) \in \mathscr{W}$ because $\rho(\bar{\varphi})^\perp(\alpha) = \rho(\bar{\varphi}\alpha)^\perp$ and $\sigma(\varphi)(\alpha) = \sigma(\varphi\alpha)$. \square

A.5 Proofs of §7

Proof (Proof of Theorem 11). Item (1) is trivial for finite contracts. The extension of the algorithm to infinite contracts is done in a standard way using a memoization context. The details can be found in the full version of [27].

As regards item (2), by a simple structural induction it is easy to establish that, given a derivation for $\mathbb{B} \vdash_k f : \sigma \leq \tau$ 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 $\mathscr{W} \stackrel{\text{def}}{=} \{(\mathbb{B}, \sigma, \tau) \mid \mathbb{B} \vdash_k f : \sigma \leq \tau\}$ is a coinductive weak k -subcontract relation. Let $(\mathbb{B}, \sigma, \tau) \in \mathscr{W}$. Then $\mathbb{B} \vdash_k f : \sigma \leq \tau$ is derivable. Let $A \stackrel{\text{def}}{=} \text{init}(f)$ and observe that $\mathbb{B} \vdash_k A$. As regards condition (1) in Definition 11, there is nothing to prove because it exactly coincides with the third premise in rule (A1). As regards condition (2), assume $\tau \xrightarrow{\varphi'}$ and $\langle \varphi, \bar{\varphi}' \rangle \in A$. From the first premise of rule (A1) we derive $\sigma \xrightarrow{\varphi}$. From the second premise we know that $\mathbb{B} \langle \varphi, \bar{\varphi}' \rangle \vdash_k f_{\langle \varphi, \bar{\varphi}' \rangle} : \sigma(\varphi) \leq \tau(\varphi')$ is derivable. We conclude $(\mathbb{B} \langle \varphi, \bar{\varphi}' \rangle, \sigma(\varphi), \tau(\varphi')) \in \mathscr{W}$ by definition of \mathscr{W} .

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

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

is a weak k -subcontract relation. Note that since g is relevant, we have that $g \xrightarrow{\langle \varphi_1, \bar{\varphi}'_1 \rangle \cdots \langle \varphi_n, \bar{\varphi}'_n \rangle}$ implies $\sigma \xrightarrow{\varphi_1 \cdots \varphi_n}$ and $\tau \xrightarrow{\varphi'_1 \cdots \varphi'_n}$. We prove that, if $(\mathbb{B}, \sigma, \tau) \in \mathcal{W}_k$, then $\mathbb{B} \vdash_k f : \sigma \preceq \tau$ is derivable by induction on the depth of σ and τ .

The base case is when both σ and τ have null depth. In this case, $A_r = A = \emptyset$ and the premises of the algorithm are trivially satisfied, since both σ and τ have just the empty ready set. In the inductive case, from $(\mathbb{B}, \sigma', \tau') \in \mathcal{W}_k$ and by definition of \mathcal{W}_k we deduce that there exist $\varphi_1, \dots, \varphi_n, \varphi'_1, \dots, \varphi'_n$ such that $g \xrightarrow{\langle \varphi_1, \bar{\varphi}'_1 \rangle \cdots \langle \varphi_n, \bar{\varphi}'_n \rangle} g'$ and $\mathbb{B} = \emptyset \langle \varphi_1, \bar{\varphi}'_1 \rangle \cdots \langle \varphi_n, \bar{\varphi}'_n \rangle$ 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 $\text{init}(g') \subseteq A_r$ in the first premise of the algorithm. Let $\langle \varphi, \bar{\varphi}' \rangle \in \text{init}(g')$. By definition of coinductive weak k -subcontract relation and from the fact that g is relevant we know that $(\mathbb{B} \langle \varphi, \bar{\varphi}' \rangle, \sigma'(\varphi), \tau'(\varphi')) \in \mathcal{W}_k$. Since $\varphi \varphi' \neq \varepsilon$, by induction hypothesis we obtain that there exists $f_{\langle \varphi, \bar{\varphi}' \rangle}$ such that $\mathbb{B} \langle \varphi, \bar{\varphi}' \rangle \vdash_k f_{\langle \varphi, \bar{\varphi}' \rangle} : \sigma'(\varphi) \preceq \tau'(\varphi')$. Hence $\text{init}(g') \subseteq A$ in the second premise of the algorithm. Since g proves $\sigma \preceq \tau$, we have that $\text{init}(g')$ satisfies condition (1) of Definition 15, which coincides with the third premise of the algorithm. From $\text{init}(g') \subseteq A$ we deduce that A also satisfies the third premise of the algorithm. Hence we can apply the rule and conclude $\mathbb{B} \vdash_k \bigvee_{\mu \in A} f_\mu : \sigma' \preceq \tau'$. The fact that the algorithm computes the best relevant orchestrator proving $\sigma \preceq \tau$ is an immediate consequence of $\text{init}(g') \subseteq A$, as shown earlier. \square