An Algebraic Theory for Web Service Contracts

Published version:

DOI:10.1007/s00165-015-0334-2

Under consideration for publication in Formal Aspects of Computing

An Algebraic Theory for Web Service Contracts

Cosimo Laneve[a,1] and Luca Padovani[b]

[a]Dipartimento di Informatica – Scienza e Ingegneria, Università di Bologna – INRIA Focus Team, Italy
[b]Dipartimento di Informatica, Università di Torino, Italy

Abstract. We study the foundations of Web service technologies for connecting abstract and concrete service definitions and for discovering services according to their observable behavior. We pursue this study addressing a subset of BPEL activities that include concurrency constructs. We present a formal semantics – called *compliance preorder* – of this subset of BPEL and we define a behavioral type discipline that guarantees the correctness of client-server interactions. The types of our discipline, called *contracts*, are De Nicola and Hennessy tau-less, finite-state CCS processes. We show that contracts are BPEL normal forms according to the compliance preorder and that the compliance preorder does coincide with a well-known equivalence in concurrency theory, the *must-testing preorder*. The compliace preorder is not fully adequate for discovering Web services though, since it does not support width and depth extensions of Web services. To address this issue, we propose a sound generalization of the compliance preorder, called *subcontract relation*, that admits a notion of principal service contract – the *dual contract* – compliant with a given client contract and that exhibits good precongruence properties when choreographies of Web services are considered.

Keywords: Web services, BPEL, contracts, compliance, must-testing, subcontract, dual contract, choreography.

1. Introduction

Service-oriented technologies and Web services have been proposed as a new way of distributing and organizing complex applications across the Internet. These technologies are nowadays extensively used for delivering cloud computing platforms. A large effort in the development of Web services has been devoted to their specification, their publication, and their use. In this context, the Business Process Execution Language for Web Services (BPEL for short) has emerged as the *de facto* standard for implementing and composing Web services and is now supported by several major software vendors (Oracle Process Manager, IBM WebSphere, and Microsoft BizTalk).

[1] The research reported in this paper was partially funded by the EU FP7 610582 project called ENVISAGE.
Correspondence and offprint requests to: Cosimo Laneve, Dipartimento di Informatica – Scienza e Ingegneria, Università di Bologna – INRIA Focus Team, Mura A. Zamboni 7, 40127 Bologna, Italy. e-mail: cosimo.laneve@unibo.it

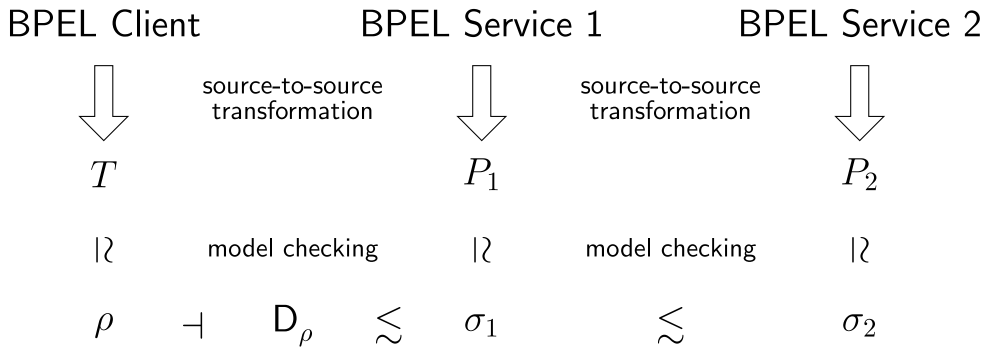

Fig. 1. Summary of contributions.

The main issue concerning the publication of Web services is the definition of appropriate service descriptions that enable their identification, discovery and composition without disclosing important details concerning their internal implementation and their binding to concrete protocols. The current standard for service description is defined by the Web Service Description Language (WSDL) [CCMW01], which specifies the format of the exchanged messages – the *schema* –, the locations where the interactions are going to occur – the *interface* –, the transfer mechanism to be used (i.e. SOAP-RPC, or others), and basic service abstractions (one-way/asynchronous and request-response/synchronous patterns of conversations). These abstractions are very simple and inadequate for expressing arbitrary, possibly cyclic protocols of exchanged messages between communicating parties. That is, the information provided by WSDL is insufficient for verifying the behavioral compliance between parties. It is also worth to notice that other technologies, such as UDDI registries (Universal Description, Discovery and Integration [BKL01]), provide limited support because registry items only include pointers to the locations of the service abstractions, without constraining the way these abstractions are defined or related to the actual implementations. In this respect, UDDI registries are almost useless for discovering services; an operation that is performed manually by service users and consumers.

The publication of abstract service descriptions, which we call *contracts*, and the related ability to discover Web services by means of their contract require studying the connection between a Web service and its contract and, more generally, defining a formal theory for reasoning about Web services by means of their contracts. In this article we provide such a theory and we do so adopting a well-known approach in concurrency theory where Web services are abstracted using process calculi and contracts are behavioral types (see, for instance, [NN94, HVK98]). More specifically, our approach is based on the following cornerstone items:

(I1) A formal semantics of BPEL to express client/service interactions.

(I2) A language of *contracts* and an algorithm that connects a Web service to its contract.

(I3) A *subcontract* relation that embodies the principle of safe Web service replacement.

We now provide a more detailed roadmap of the overall approach, schematically illustrated in Figure 1, and how it unfolds in the rest of the article.

(I1) BPEL abstract activities. In Section 2 we identify a sublanguage of BPEL that captures its concurrency and communication constructs, called BPEL *abstract activities*, and we ignore the details related to the actual syntax of BPEL, the schema and content of messages, and the definition of transmission protocols. These aspects are largely orthogonal to our investigation. We do not commit to a particular interpretation of actions occurring in abstract activities either: they may represent different typed channels, different operations, different types of messages. Abstract activities are denoted as the terms T, P_1, and P_2 in Figure 1 and represent *over-approximations* of the behavior of the corresponding concrete BPEL processes. By this we mean that every action that can be executed by a concrete BPEL process can also be executed by its corresponding abstract activity, but the converse does not necessarily hold. Such over-approximation is necessary in order for the theory to be decidable and tractable.

The semantics of BPEL abstract activities is defined by a technique based on tests [DH84]. That is:

- we define a basic observation, called *compliance* and noted $T \dashv P$, which holds whenever T successfully completes *every* interaction with P; here "successfully" means that T never gets stuck (this notion is purposefully asymmetric as client's satisfaction is our main concern).
- we derive a *compliance preorder* by comparing the sets of clients that successfully interact with BPEL abstract activities: two BPEL abstract activities are equivalent if they satisfy the same clients.

The assumption to work with over-approximations of BPEL processes means that, if T successfully interacts with P and T and P are over-approximations of the BPEL processes C and S respectively, then C successfully interacts with S as well. Obviously, since our notion of compliance between abstract activities solely concerns the communication aspects of the interaction and not, for example, the actual content of messages, the meaning of "successful interaction" should be interpreted in this more abstract scope.

(I2) Contracts. In Section 3 we define contracts as the sub-calculus of De Nicola and Hennessy tau-less CCS [DH87] consisting of prefixing, internal and external choices, and recursion. We demonstrate that our contracts, noted τ, σ_1, and σ_2 in Figure 1, retain convenient properties:

1. contracts do not disclose implementations details of BPEL processes;
2. contracts have finite-state models;
3. contracts are normal forms of BPEL abstract activities with respect to compliance preorder ($\simeq$ in Figure 1).

A consequence of properties 2 and 3 is that BPEL abstract activities also have finite-state models. It is therefore possible to develop algorithms for connecting abstract activities with their contract. In addition, in Section 4 we show that the compliance preorder corresponds to a well-known semantics in concurrency theory, the *must-testing semantics*. This means that the whole plethora of algorithms and tools already developed for must-testing theories, such as the Concurrency Workbench [CPS93], can be applied to our framework for reasoning on contracts and BPEL abstract activities. It should be remarked that the connection between the compliance preorder and must-testing is not obvious, since the two relations are induced by tests with quite different features. To prove their equivalence, we define an alternative semantics of must-testing formulated in a coinductive way, which supports a powerful proof technique.

(I3) Subcontract relation. In Section 5 we observe that the compliance preorder is a fine-grained semantics of BPEL activities that forbids two key properties that are useful for service discovery. These properties are called *width* and *depth extension*. By width extension we mean the replacement of a service with another one that provides new functionalities (called operations, in the Web service terminology); by depth extension we mean the replacement of a service with another one that allows for longer communications beyond the terminal states of the original service. We therefore define a variant of the compliance preorder, called *subcontract preorder* and noted $\lesssim$ in Figure 1, which supports these forms of extensions. Notwithstanding the differences in the corresponding preorder relations, the equivalences induced by $\lesssim$ and $\approx$ do coincide. This means that, if a client is subcontract-compliant with a contract σ_1, then it will be subcontract-compliant with the corresponding abstract BPEL activity P_1, as well as with every activity P_2 that (width/depth-) extends P_1.

We then analyze the problem of querying a repository of BPEL activities. In Section 6, we define an algorithm that takes a client T exposing a certain behavior ρ and returns *the smallest service contract* (according to the subcontract preorder) that satisfies the client – the *dual contract*, noted D_ρ in Figure 1. This contract, acting like a *principal type* in type systems, guarantees that a query to a Web service registry is answered with the largest possible set of compatible services in the registry's databases.

As a validation step for our theory, in Section 7 we show that the subcontract relation is well behaved when applied to choreographies of Web services [KBR$^+$05]. Technically, this means that $\lesssim$ is a pre-congruence with respect to parallel composition under mild conditions. This property has important practical consequences, since it enables the modular refinement of complex systems.

We conclude with a discussion of related works in Section 8, and a summary of contributions and directions for future research in Section 9. Proofs of the results are deferred to Appendix A.

Origin of the material. The basic ideas of this article have appeared in conference proceedings. In particular, the theory of contracts we use is introduced in [LP07] while the relation between (abstract) BPEL activities and contracts has been explored in [LP13]. This article is a thoroughly revised and enhanced version of [LP07, LP13] that presents the whole framework in a uniform setting and includes the full proofs of all the results. A more detailed comparison with other related work is deferred to Section 8.

2. BPEL Abstract Activities

In this section we define a model of BPEL processes that is suitable to be formally investigated. The idea is to over-approximate the behavior of BPEL processes using terms of a process algebra in such a way that the actual interacting behavior of a BPEL process is one of the possible interacting behaviors expressed by the corresponding term. We will not be able to formally prove that the approximation we provide is sound, since BPEL is not equipped with a formal semantics. We will nonetheless argue in favor of this property.

2.1. A Quick Look at BPEL

In BPEL, business processes are described as the composition of basic activities, which include the sending and receiving of messages. We introduce the basic notions of BPEL looking at a stripped off version of the initial business process example in the language specification [Alv07]. The XML document in Figure 2 describes the behavior of an e-commerce service that interacts with four other partners, one of them being the customer (identified by the name `purchasing` in the figure), the other ones being a service (identified by `invoicing`) that provides prices, a service (identified by `shipping`) that takes care of the shipment of goods, and a service (identified by `scheduling`) that schedules the manufacturing of goods. The business process is made of *activities*, which can be either *atomic* or *composite*. In this example atomic activities consist of the *invocation* of operations in other partners (lines 10–14, 22–27, 31–36), the *acceptance* of messages from other partners, either as incoming requests (line 3) or as responses to previous invocations (lines 15–19 and 28), and the *sending* of responses to clients (line 39). Atomic activities are composed together into so-called structured activities, such as *sequential composition* (see the `sequence` fragments) and *parallel composition* (see the `flow` fragment at lines 4–38). In a `sequence` fragment, all the child activities are executed in the order in which they appear, and each activity begins the execution only after the previous one has completed. In a `flow` fragment, all the child activities are executed in parallel, and the whole `flow` activity completes as soon as *all* the child activities have completed. It is possible to constrain the execution of parallel activities by means of *links*. In the example, there is a link `ship-to-invoice` declared at line 6 and used in lines 12 and 25, meaning that the invocation at lines 23–27 cannot take place before the one at lines 10–14 has completed. Similarly, the link `ship-to-scheduling` means that the invocation at lines 32–36 cannot take place before the receive operation at lines 15–19 has completed. In short, the presence of links limits the possible interleaving of the activities in a `flow` fragment.

BPEL includes other conventional constructs not shown in the example, such as conditional and iterative execution of activities. For example, the BPEL activity

```
<if>
  <condition> bool-expr </condition>
  activity-True
  <else> activity-False </else>
</if>
```

evaluates `bool-expr`, which must be a Boolean condition, and executes either `activity-True` or `activity-False` depending on whether the condition turns out to be true or false. Similarly, the activity

```
<while>
  <condition> bool-expr </condition>
  activity
</while>
```

specifies that `activity` should be repeatedly executed as long as the Boolean condition `bool-expr` is true.

```
<process>
  <sequence>
    <receive partnerLink="purchasing" operation="sendPurchaseOrder"/>
    <flow>
      <links>
        <link name="ship-to-invoice"/>
        <link name="ship-to-scheduling"/>
      </links>
      <sequence>
        <invoke partnerLink="shipping" operation="requestShipping">
          <sources>
            <source linkName="ship-to-invoice"/>
          </sources>
        </invoke>
        <receive partnerLink="shipping" operation="sendSchedule">
          <sources>
            <source linkName="ship-to-scheduling"/>
          </sources>
        </receive>
      </sequence>
      <sequence>
        <invoke partnerLink="invoicing" operation="initiatePriceCalculation"/>
        <invoke partnerLink="invoicing" operation="sendShippingPrice">
          <targets>
            <target linkName="ship-to-invoice"/>
          </targets>
        </invoke>
        <receive partnerLink="invoicing" operation="sendInvoice"/>
      </sequence>
      <sequence>
        <invoke partnerLink="scheduling" operation="requestProductionScheduling"/>
        <invoke partnerLink="scheduling" operation="sendShippingSchedule">
          <targets>
            <target linkName="ship-to-scheduling"/>
          </targets>
        </invoke>
      </sequence>
    </flow>
    <reply partnerLink="purchasing" operation="sendPurchaseOrder"/>
  </sequence>
</process>
```

Fig. 2. BPEL business process for an e-commerce service.

2.2. A Formal Model of BPEL Abstract Activities

To pursue our formal investigation, we will now present an abstract language of activities whose operators correspond to those found in BPEL.

We use a set N of *names*, ranged over by $a, b, c, \dots$, that represent communication channels or message types and a disjoint set $\overline{\mathsf{N}}$ of *co-names*, ranged over by $\overline{a}, \overline{b}, \overline{c}, \dots$; the term *action* refers to names and co-names without distinction; actions are ranged over by $\alpha, \beta, \dots$. We use A, B, $\dots$ to range over sets of names and we define an involution $\overline{\cdot}$ such that $\overline{\overline{a}} = a$. We use $\varphi, \psi, \dots$ to range over $(\mathsf{N} \cup \overline{\mathsf{N}})^*$ and $\mathrm{R}, \mathrm{S}, \dots$ to range over finite sets of actions. Let $\overline{\mathrm{R}} \stackrel{\text{def}}{=} \{\overline{\alpha} \mid \alpha \in \mathrm{R}\}$.

The syntax of BPEL *abstract activities* is defined by the grammar in Table 1, where each construct has been named after the corresponding XML tag in BPEL. Essentially we represent BPEL abstract activities as terms

Table 1. Syntax of BPEL abstract activities.

$$
\begin{array}{rcll}
P, Q, P_i & ::= & \mathtt{0} & (\mathtt{empty}) \\
& | & a & (\mathtt{receive}) \\
& | & \overline{a} & (\mathtt{invoke}) \\
& | & \sum_{i\in I} \alpha_i \,;\, P_i & (\mathtt{pick}) \\
& | & P \mid_A Q & (\mathtt{flow\ \&\ link}) \\
& | & P \,;\, Q & (\mathtt{sequence}) \\
& | & \bigoplus_{i\in I} P_i & (\mathtt{if}) \\
& | & P^* & (\mathtt{while})
\end{array}
$$

of a simple process algebra similar to Milner's CCS [Mil82] and Hoare's CSP [BHR84]. We are interested in the interactions of BPEL activities with the external environment rather than in the actual implementation of business processes. For this reason, our process language overlooks details regarding internal, unobservable computations, exception handling, value passing and focuses on the communication behavior of activities.

The activity `0` represents the completed process that performs no actions. The activity a represents the act of waiting for an incoming message. Here we take the point of view that a stands for a particular operation implemented by the process. The activity $\overline{a}$ represents the act of invoking the operation a provided by another partner. The activity $\sum_{i\in I} \alpha_i \,;\, P_i$ represents the act of waiting for any of the α_i operations to be performed, i belonging to a *finite* set I. Whichever operation α_i is performed, it first disables the remaining ones and the continuation P_i is executed. If $\alpha_i = \alpha_j$ and $i \neq j$, then the choice whether executing P_i or P_j is implementation dependent. The process $P \mid_A Q$, where A is a set of names, represents the parallel composition (`flow`) of P and Q and the creation of a private set A of `link` names that will be used by P and Q to synchronize; an example will be given shortly. The n-ary version $\prod_{i\in 1..n}^{A} P_i$ of this construct may also be considered: we stick to the binary one for simplicity. The process $P \,;\, Q$ represents the sequential composition of P followed by Q. Again we only provide a binary operator, where the BPEL one is n-ary. The process $\bigoplus_{i\in I} P_i$ represents an internal choice performed by the process, that results into one of the finite I continuations P_i. Finally, P^* represents the repetitive execution of process P so long as an internally verified condition is satisfied.

The `pick` activity $\sum_{i\in 1..n} \alpha_i \,;\, P_i$ and the `if` activity $\bigoplus_{i\in 1..n} P_i$ will also be written $\alpha_1 \,;\, P_1 + \cdots + \alpha_n \,;\, P_n$ and $P_1 \oplus \cdots \oplus P_n$, respectively. In the following we treat (`empty`), (`receive`), and (`invoke`) as special cases of (`pick`), while at the same time keeping the formal semantics just as easy. In particular, we write `0` for $\sum_{\alpha\in\varnothing} \alpha \,;\, P_\alpha$ and α as an abbreviation for $\sum_{\beta\in\{\alpha\}} \beta \,;\, \mathtt{0}$ (tailing `0` are always omitted).

As we have anticipated, the language omits the details about the conditions that determine which branch of an `if` is taken or how many times an activity is iterated. For example, the `if` activity shown at the end of Section 2.1 will be abstracted into the process $\mathtt{activity\text{-}True} \oplus \mathtt{activity\text{-}False}$, meaning that one of the two activities will be performed and the choice will be a consequence of some unspecified internal decision. A similar observation pertains to the `<while>` activity (see also Remark 2.2).

Example 2.1. *The* BPEL *activity in Figure 2 can be described by the term below, where for the sake of readability we give short names to the operations used in the activity as by Table 2:*

$$
\mathtt{sPO} \,;\, \Big(\overline{\mathtt{rS}} \,;\, \big((\overline{\mathtt{sti}} \mid_\varnothing \mathtt{sS} \,;\, \overline{\mathtt{sts}}) \mid_{\{\mathtt{sti}\}} \overline{\mathtt{iPC}} \,;\, \mathtt{sti} \,;\, \overline{\mathtt{sSP}} \,;\, \mathtt{sI}\big) \mid_{\{\mathtt{sts}\}} \overline{\mathtt{rPS}} \,;\, \mathtt{sts} \,;\, \overline{\mathtt{sSS}}\Big) \,;\, \overline{\mathtt{sPO}} \tag{1}
$$

Note that we use names for specifying both actions and links. For example, we represent the source of the link `ship-to-invoice` *as the action* $\overline{\mathtt{sti}}$ *and the corresponding target as the action* `sti`*. Since* `sti` *guards the actions* $\overline{\mathtt{sSP}}$ *and* `sI`*, these will not be executed until after* $\overline{\mathtt{rS}}$*, which guards* $\overline{\mathtt{sti}}$*, has been executed. Similarly for the* `ship-to-scheduling` *link. Note that names corresponding to links are restricted so that they are not visible from outside. Indeed, we will see that they do not appear in activity's behavioral description.* ■

Remark 2.1. *The* BPEL *specification defines a number of static analysis requirements beyond the mere syntactic correctness of processes whose purpose is to "detect any undefined semantics or invalid semantics within a process definition" [Alv07]. Several of these requirements regard the use of links. For example, it is required that no link must cross the boundary of a repeatable construct (*`while`*). It is also required that*

Table 2. Legend for the operations of the BPEL process in Figure 2.

Name	Operation
`sPO`	`sendPurchaseOrder`
`rS`	`requestShipping`
`sS`	`sendSchedule`
`iPC`	`initiatePriceCalculation`
`sSP`	`sendShippingPrice`
`sI`	`sendInvoice`
`rPS`	`requestProductionScheduling`
`sSS`	`sendShippingSchedule`
`sti`	`ship-to-invoice`
`sts`	`ship-to-scheduling`

link ends must be used exactly once (hence $0 \,|_{\{a\}}\, a$ is invalid because $\overline{a}$ is never used), and the dependency graph determined by links must be acyclic (hence $a.\overline{b} \,|_{\{a,b\}}\, b.\overline{a}$ is invalid because it contains cycles). These constraints may be implemented by restricting the arguments to the above abstract activities and then using static analysis techniques. ■

2.3. Operational Semantics of BPEL Abstract Activities

In order to reason about abstract activities, the language in Table 1 must be equipped with semantics that have both a sensible discriminating power and some convenient proof techniques. The usual approach in concurrency theory is to define a transition relation that represents process evolution and to define an observation predicate that detects the successful termination [Hen88, Mil89]. Based on these two notions, one derives the semantics of a process by testing the observation predicate under all possible contexts.

The operational semantics of BPEL abstract activities is defined in Table 3. In the table we define two relations: $P\checkmark$, read P *has completed*, and $P \xrightarrow{\mu} Q$, where μ ranges over actions and the special name ε denoting internal computations, as the least ones satisfying the corresponding rules. The table does not report the symmetric rules for $|$.

According to Table 3, the process $\sum_{i\in I} \alpha_i \,;\, P_i$ has as many α-labelled transitions as the number of actions in $\{\alpha_i \mid i \in I\}$. After a visible transition, only the selected continuation is allowed to execute. The process $\bigoplus_{i\in I} P_i$ may internally choose to behave as one of the P_i, with $i \in I$. The process $P \,|_A\, Q$ allows P and Q to internally evolve autonomously, or to emit/receive messages on names not in the set A, or to synchronize with each other on names in A. It completes when both P and Q have completed. The process $P \,;\, Q$ reduces according to the reductions of P first, and of Q when P has completed. Finally, the process P^* may either complete in one step by reducing to 0, or it may execute P one more time followed by P^*. The choice among the two possibilities results from an internal computation which is left implicit in the model.

Remark 2.2. *According to the operational semantics, P^* may execute the activity P an arbitrary number of times. This is at odds with concrete* BPEL *activities having P^* as abstract counterpart. For example, in* BPEL *it is possible to write a process like*

```
<while>
   <condition> bool-expr </condition>
   activity
</while>
```

which means executing `activity` *as long as the* `bool-expr` *condition is true. Representing such* BPEL *activity with* $\texttt{activity}^*$ *means* over-approximating *it: the above fragment of* BPEL *executes* $\texttt{activity}^n$ *for an arbitrary n; we approximate this as* $\texttt{activity}^*$ *which corresponds to* $0 \oplus \texttt{activity} \oplus \texttt{activity}^2 \oplus \cdots$. *This approximation is crucial for Lemma 2.1 below.* ■

We illustrate the semantics of BPEL abstract activities through few examples:

Table 3. Operational semantics of abstract BPEL

Completion predicate $\boxed{P\checkmark}$

$$\mathsf{0}\checkmark \qquad \frac{P\checkmark \quad Q\checkmark}{P \mid_A Q\checkmark} \qquad \frac{P\checkmark \quad Q\checkmark}{P\,;Q\checkmark}$$

Transition relation $\boxed{P \xrightarrow{\mu} Q}$

$$\text{(ACTION)}\ \sum_{i\in I} \alpha_i\,;P_i \xrightarrow{\alpha_i} P_i \qquad \text{(IF)}\ \bigoplus_{i\in I} P_i \xrightarrow{\varepsilon} P_i$$

$$\text{(FLOW)}\ \frac{P \xrightarrow{\mu} P' \quad \mu \notin A \cup \overline{A}}{P \mid_A Q \xrightarrow{\mu} P' \mid_A Q} \qquad \text{(LINK)}\ \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q' \quad \alpha \in A \cup \overline{A}}{P \mid_A Q \xrightarrow{\varepsilon} P' \mid_A Q'}$$

$$\text{(SEQ)}\ \frac{P \xrightarrow{\mu} P'}{P\,;Q \xrightarrow{\mu} P'\,;Q} \qquad \text{(SEQ-END)}\ \frac{P\checkmark \quad Q \xrightarrow{\mu} Q'}{P\,;Q \xrightarrow{\mu} Q'} \qquad \text{(WHILE-END)}\ P^* \xrightarrow{\varepsilon} \mathsf{0} \qquad \text{(WHILE)}\ \frac{P \xrightarrow{\mu} P'}{P^* \xrightarrow{\mu} P'\,;P^*}$$

1. $(\overline{a} \oplus \overline{b} \mid_{\{a,b\}} a \oplus b)\,;\overline{c} \xrightarrow{\varepsilon} (\overline{a} \mid_{\{a,b\}} a \oplus b)\,;\overline{c}$ by (IF), (FLOW), and (SEQ). By the same rules, it is possible to have $(\overline{a} \mid_{\{a,b\}} a \oplus b)\,;\overline{c} \xrightarrow{\varepsilon} (\overline{a} \mid_{\{a,b\}} b)\,;\overline{c}$, which cannot reduce anymore ($\overline{a} \mid_{\{a,b\}} b$ is a *deadlocked* activity).
2. let $\Psi \stackrel{\text{def}}{=} \mathsf{0}\,;(\mathsf{0} \oplus \mathsf{0})^*$. Then, according to rules (SEQ-END), (IF), and (WHILE), $\Psi \xrightarrow{\varepsilon} \Psi$ and $\Psi \xrightarrow{\varepsilon} \mathsf{0}$.
3. $(\overline{a} \mid_{\{a\}} a)^* \xrightarrow{\varepsilon} \mathsf{0} \mid_{\{a\}} \mathsf{0}\,;(\overline{a} \mid_{\{a\}} a)^*$ by rules (LINK) and (WHILE).

In the following we write $\stackrel{\varepsilon}{\Longrightarrow}$ for the reflexive, transitive closure of $\xrightarrow{\varepsilon}$ and $\stackrel{\alpha}{\Longrightarrow}$ for the composition $\stackrel{\varepsilon}{\Longrightarrow}\xrightarrow{\alpha}\stackrel{\varepsilon}{\Longrightarrow}$; we also write $P \xrightarrow{\mu}$ (respectively, $P \stackrel{\alpha}{\Longrightarrow}$) if there exists Q such that $P \xrightarrow{\mu} Q$ (respectively, $P \stackrel{\alpha}{\Longrightarrow} Q$); we let $P \stackrel{\mu}{\not\longrightarrow}$ if not $P \xrightarrow{\mu}$.

A relevant property of our BPEL abstract calculus is that the model of every activity P, that is the set of processes reachable from P by means of arbitrary reductions, is always finite. Because of this, it is possible to devise verification techniques of activities by reasoning directly on the models, rather their abstraction.

Lemma 2.1. *Let* $\mathsf{reach}(P) \stackrel{\text{def}}{=} \{Q \mid \exists\varphi : P \stackrel{\varphi}{\Longrightarrow} Q\}$. *Then, for every activity P, the set* $\mathsf{reach}(P)$ *is finite.*

We introduce a number of auxiliary definitions that will be useful in the rest of the paper. By Lemma 2.1 these notions are trivially decidable.

Definition 2.1. *We introduce the following notation:*

- *We say that P* diverges*, notation $P\uparrow$, if there is an infinite sequence of ε-transitions $P \xrightarrow{\varepsilon}\xrightarrow{\varepsilon} \cdots$ starting from P. We say that P* converges*, notation $P\downarrow$, if it does not diverge.*
- *We let* $\mathsf{init}(P) \stackrel{\text{def}}{=} \{\alpha \mid P \stackrel{\alpha}{\Longrightarrow}\}$ *be the set of* initial visible actions *performed by P.*
- *We say that P has* ready set R*, notation $P \Downarrow \mathrm{R}$, if $P \stackrel{\varepsilon}{\Longrightarrow} Q$ and* $\mathrm{R} = \mathsf{init}(Q)$.
- *Let $P \stackrel{\alpha}{\Longrightarrow}$. Then* $P(\alpha) \stackrel{\text{def}}{=} \bigoplus_{P \stackrel{\varepsilon}{\Longrightarrow}\xrightarrow{\alpha} Q} Q$. *We call $P(\alpha)$ the* continuation of P after α.

These definitions are almost standard, except for $P(\alpha)$ (that we already used in [LP07]). The abstract activity $P(\alpha)$ represents the residual behavior of P after an action α, from the point of view of the party that is interacting with P. Indeed, the party does not know which, of the possibly multiple, α-labelled branches P has taken. For example $(a\,;b + a\,;c + b\,;d)(a) = b \oplus c$ and $(a\,;b + a\,;c + b\,;d)(b) = d$.

2.4. The Compliance Preorder

We proceed defining a notion of equivalence between abstract activities that is based on their observable behavior. To this aim, we introduce a special name e (not in N) for denoting the successful termination of an abstract activity ("e" stands for end). We let T range over *client* activities, that is activities that may contain such special name e. By *compliance* between a "client" activity T and a "service" activity P we mean that every interaction between T and P, where P stops communicating with T, is such that T has reached a successfully terminated state. Following De Nicola and Hennessy's approach to process semantics [DH84], this compliance relation induces a preorder on services on the basis of the set of client activities that comply with a given service activity.

Definition 2.2 (Compliance)**.** *The (client) activity T is* compliant with *the (service) activity P, written $T \dashv P$, if $P \mid_{\mathsf{N}} T \stackrel{\varepsilon}{\Longrightarrow} P' \mid_{\mathsf{N}} T'$ implies:*

1. *if $P' \mid_{\mathsf{N}} T' \stackrel{\varepsilon}{\not\longrightarrow}$, then $\{\mathsf{e}\} \subseteq \mathsf{init}(T')$, and*
2. *if $P'\!\uparrow$, then $\{\mathsf{e}\} = \mathsf{init}(T')$.*

The compliance preorder *is the relation induced by compliance: $P \sqsubseteq Q$ if and only if $T \dashv P$ implies $T \dashv Q$ for every T. We write $\eqsim$ for $\sqsubseteq \cap \sqsupseteq$.*

According to the notion of compliance, if the client-service conversation terminates, then the client is in a successful state (it will emit an e-name). For example, $a ; \mathsf{e} + b ; \mathsf{e} \dashv \overline{a} \oplus \overline{b}$ and $a ; \mathsf{e} \oplus b ; \mathsf{e} \dashv \overline{a} + \overline{b}$ but $a ; \mathsf{e} \oplus b ; \mathsf{e} \not\dashv \overline{a} \oplus \overline{b}$ because of the computation $\overline{a} \oplus \overline{b} \mid_{\mathsf{N}} a ; \mathsf{e} \oplus b ; \mathsf{e} \stackrel{\varepsilon}{\Longrightarrow} \overline{b} \mid_{\mathsf{N}} a ; \mathsf{e} \stackrel{\varepsilon}{\not\longrightarrow}$ where the client waits for an interaction on a in vain. Similarly, the client must reach a successful state if the conversation does not terminate but the divergence is due to the service. In this case, however, every reachable state of the client must be such that the only possible action is e. The practical justification of such a notion of compliance derives from the fact that connection-oriented communication protocols (like those used for interaction with Web services) typically provide for an explicit end-of-connection signal. Consider for example the client behavior $\mathsf{e} + \overline{a} ; \mathsf{e}$. Intuitively this client tries to send a request on the name a, but it can also succeed if the service rejects the request. So $\mathsf{e} + \overline{a} ; \mathsf{e} \dashv \mathsf{0}$ because the client can detect the fact that the service is not ready to interact on a. The same client interacting with a diverging service would have no way to distinguish a service that is taking a long time to accept the request from a service that is perpetually performing internal computations, hence $\mathsf{e} + \overline{a} ; \mathsf{e} \not\dashv \Psi$. As a matter of fact, the definition of compliance makes Ψ the "smallest service" – the one a client can make the least number of assumptions on (this property will be fundamental in the definition of principal dual contract in Section 6). That is $\Psi \sqsubseteq P$, for every P. As another example, we notice that $a ; b + a ; c \sqsubseteq a ; (b \oplus c)$ since, after interacting on a, a client of the smaller service is not aware of which state the service is in (it can be either b or c).

Example 2.2. *As a counter-example of compliance, consider the process*

$$\mathtt{sPO} ; \overline{\mathtt{rS}} ; \big((\mathtt{sS} \mid_{\varnothing} \overline{\mathtt{rPS}}) ; \overline{\mathtt{sSS}} \mid_{\varnothing} \overline{\mathtt{iPC}} ; \overline{\mathtt{sSP}} ; \mathtt{sI} \big) ; \overline{\mathtt{sPO}} \tag{2}$$

which has been obtained from Example 2.1 by removing and serializing the synchronizations described by the links. It is relevant to ask whether this implementation of the e-commerce service is equivalent to the previous one according to the compliance pre-order. It turns out that this is not the case, in particular the client activity

$$\overline{\mathtt{sPO}} ; (\mathsf{e} + \mathtt{rPS})$$

is compliant with (2) *but not with* (1)*, while the client activity*

$$\overline{\mathtt{sPO}} ; \mathtt{rPS} ; \mathsf{e}$$

is compliant with (1) *but not with* (2)*. For example, after the two operations* sPO *and* rPS*, the first test reduces to* 0*, which allows no further synchronizations with the service and does not perform* e *actions. It can be shown that* (1) *is compliant-equivalent to the abstract activity*

$$\mathtt{sPO} ; \Big(\overline{\mathtt{rS}} ; \big((\mathtt{sS} \mid_{\varnothing} \overline{\mathtt{rPS}}) ; \overline{\mathtt{sSS}} \mid_{\varnothing} \overline{\mathtt{iPC}} ; \overline{\mathtt{sSP}} ; \mathtt{sI} \big) + \overline{\mathtt{rPS}} ; \overline{\mathtt{rS}} ; (\mathtt{sS} ; \overline{\mathtt{sSS}} \mid_{\varnothing} \overline{\mathtt{iPC}} ; \overline{\mathtt{sSP}} ; \mathtt{sI}) \Big) ; \overline{\mathtt{sPO}} \qquad \blacksquare$$

As by Definition 2.2, it is difficult to formally show the compliance preorder between two activities because of the universal quantification over all (client) activities T. For this reason, in Section 4, we will provide an alternative characterization of $\sqsubseteq$ that allows us to prove the compliance preorder without any universal quantification.

3. Contracts

Following the longstanding approach of behavioral type systems that enforce correctness invariants on interactions of concurrent systems [HVK98], in this section we discuss how to associate abstract descriptions, called *contracts*, to a BPEL abstract activity. There is always a tradeoff between detail and abstraction when defining a contract language. In general, three criteria should be taken in consideration:

(1) contracts should be expressive enough to enable reasoning about the compliance of BPEL activities;
(2) contracts, being public, should not disclose the internal structure and actual implementation of services;
(3) contracts, like behavioral types, should support automated model/type checking tools that associate them with processes.

We consider a set of *contract names*, ranged over $\mathsf{C}, \mathsf{C}', \mathsf{C}_1, \dots$. A *contract* is a tuple

$$(\mathsf{C}_1 = \sigma_1, \dots, \mathsf{C}_n = \sigma_n,\ \sigma)$$

where $\mathsf{C}_i = \sigma_i$ are *contract name definitions*, σ is the main term, and we assume that there is no chain of definitions of the form $\mathsf{C}_{n_1} = \mathsf{C}_{n_2}$, $\mathsf{C}_{n_2} = \mathsf{C}_{n_3}$, $\dots$, $\mathsf{C}_{n_k} = \mathsf{C}_{n_1}$. The syntax of the σ_i's and of σ is given by the grammar below:

$$\sigma \quad ::= \quad \mathsf{C} \ \mid\ \alpha\,;\sigma \ \mid\ \sigma + \sigma \ \mid\ \sigma \oplus \sigma$$

where $\mathsf{C} \in \{\mathsf{C}_1, \dots, \mathsf{C}_n\}$. The contract $\alpha\,;\sigma$ represents sequential composition in the restricted form of prefixing. The operators $+$ and $\oplus$, referred to as *external* and *internal* choice, correspond to `pick` and `if` of BPEL activities, respectively. These operations are assumed to be associative and commutative; therefore we will write $\sigma_1 + \cdots + \sigma_n$ and $\sigma_1 \oplus \cdots \oplus \sigma_n$ without confusion and will sometimes shorten these contracts as $\sum_{i \in 1..n} \sigma_i$ and $\bigoplus_{i \in 1..n} \sigma_i$, respectively. The contract name C is used to model recursive behaviors such as $\mathsf{C} = a\,;\mathsf{C}$. In what follows we will leave contract name definitions implicit and identify a contract $(\mathsf{C}_1 = \sigma_1, \dots, \mathsf{C}_n = \sigma_n,\ \sigma)$ with its main body σ. We will write $\mathsf{cnames}(\sigma)$ for the set $\{\mathsf{C}_1, \dots, \mathsf{C}_n\}$ and $\mathsf{actions}(\sigma)$ for the set of actions occurring in σ or in any of the σ_i.

The operational semantics of contracts is defined by the rules below:

$$\alpha\,;\sigma \xrightarrow{\alpha} \sigma \qquad \sigma \oplus \rho \xrightarrow{\varepsilon} \sigma \qquad \frac{\sigma \xrightarrow{\varepsilon} \sigma'}{\sigma + \rho \xrightarrow{\varepsilon} \sigma' + \rho} \qquad \frac{\sigma \xrightarrow{\alpha} \sigma'}{\sigma + \rho \xrightarrow{\alpha} \sigma'} \qquad \frac{\mathsf{C} = \sigma \quad \sigma \xrightarrow{\mu} \sigma'}{\mathsf{C} \xrightarrow{\mu} \sigma'}$$

plus the symmetric of rules $+$ and $\oplus$. Note that $+$ evaluates the branches as long as they can perform invisible actions. This rule is absent in BPEL abstract activities because, there, the branches are always guarded by an action.

In the following we will use these definitions:

- $\mathsf{0} \stackrel{\text{def}}{=} \mathsf{C}_0$, where $\mathsf{C}_0 = \mathsf{C}_0 + \mathsf{C}_0$ represents a terminated activity;
- $\Omega \stackrel{\text{def}}{=} \mathsf{C}_\Omega$, where $\mathsf{C}_\Omega = \mathsf{C}_\Omega \oplus \mathsf{C}_\Omega$ represents divergence, that is a non-terminating activity.

In particular, there are no μ and σ such that $\mathsf{0} \xrightarrow{\mu} \sigma$ and $\Omega \xrightarrow{\varepsilon} \Omega$ is the only transition of Ω. Although the contract language is apparently simpler than BPEL abstract activities, it *is not* a sublanguage of the latter. In fact, Ω cannot be written as a term in the syntax of Section 2. Nevertheless, in the following we will demonstrate that contracts provide alternative descriptions (with respect to the preorder $\sqsubseteq$) to BPEL abstract activities.

We can relate BPEL abstract activities and contracts by means of the corresponding transition systems.

```
<process>
  <sequence>
    <receive partnerLink="e-commerce" operation="Login"/>
    <while>
      <condition>
        ... check credentials ...
      </condition>
      <sequence>
        <invoke partnerLink="e-commerce" operation="InvalidLogin"/>
        <receive partnerLink="e-commerce" operation="Login"/>
      </sequence>
    </while>
    <invoke partnerLink="e-commerce" operation="ValidLogin"/>
    ...
  </sequence>
</process>
```

Fig. 3. BPEL business process for an e-commerce service.

To this aim, let X and Y range over BPEL abstract activities *and* contracts. Then, X and Y interact according to the rules

$$\frac{\mathsf{X} \xrightarrow{\mu} \mathsf{X}' \quad \mu \notin A \cup \overline{A}}{\mathsf{X} \,|_A\, \mathsf{Y} \xrightarrow{\mu} \mathsf{X}' \,|_A\, \mathsf{Y}} \qquad \frac{\mathsf{Y} \xrightarrow{\mu} \mathsf{Y}' \quad \mu \notin A \cup \overline{A}}{\mathsf{X} \,|_A\, \mathsf{Y} \xrightarrow{\mu} \mathsf{X} \,|_A\, \mathsf{Y}'} \qquad \frac{\mathsf{X} \xrightarrow{\alpha} \mathsf{X}' \quad \mathsf{Y} \xrightarrow{\overline{\alpha}} \mathsf{Y}' \quad \alpha \in A \cup \overline{A}}{\mathsf{X} \,|_A\, \mathsf{Y} \xrightarrow{\varepsilon} \mathsf{X}' \,|_A\, \mathsf{Y}'}$$

It is possible to extend the definition of compliance to contracts and, by Definition 2.2, obtain a relation that allows us to compare activities and contracts without distinction. To be precise, the relation $\mathsf{X} \sqsubseteq \mathsf{Y}$ is smaller (in principle) than the relation $\sqsubseteq$ given in Definition 2.2 because, as we have said, the contract language is not a sublanguage of that of activities and, therefore, the set of tests that can be used for comparing X and Y is larger. Nonetheless, in Section 4, we demonstrate that $\sqsubseteq$ of Definition 2.2 coincides with the relation $\mathsf{X} \sqsubseteq \mathsf{Y}$. This is a key point of our development, which will allow us to safely use the same symbol $\sqsubseteq$ for both languages and to define, for every activity P, a contract σ_P such that $P \approx \sigma_P$. In particular, we let C_P be the contract name defined by

$$\mathsf{C}_P = \begin{cases} \Omega & \text{if } P\uparrow \\ \bigoplus_{P \Downarrow \mathrm{R}} \sum_{\alpha \in \mathrm{R}} \alpha \,;\, \mathsf{C}_{P(\alpha)} & \text{otherwise} \end{cases}$$

Intuitively, when P diverges, the contract C_P associated with P is the canonical diverging contract Ω. When P converges, then C_P has as many top-level states as the ready sets of P, which are in correspondence with all the residuals to which P may reduce by means of invisible moves. For each ready set R of P, the contract of P exposes all and only the visible actions α in R and continues as $\mathsf{C}_{P(\alpha)}$. We illustrate the computation of C_P by means of an example.

Example 3.1. *Figure 3 reports the initial fragment of the* BPEL *code that implements the e-commerce service whose conversation is shown in Figure 4 and is discussed in Example 3.1. The e-commerce service is represented in abstract* BPEL *as the process P defined by*

$$P \stackrel{\text{def}}{=} \texttt{Login} \,;\, (\overline{\texttt{InvalidLogin}} \,;\, \texttt{Login})^{*} \,;\, \overline{\texttt{ValidLogin}} \,;\, Q$$

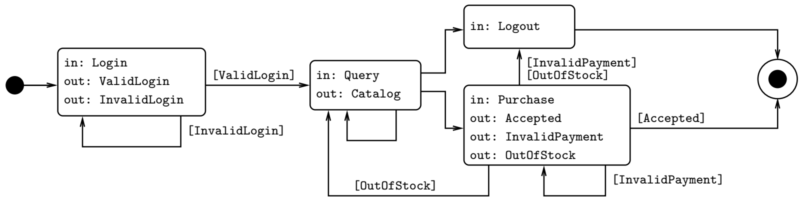

Fig. 4. Contract of a simple e-commerce service as a WSCL diagram.

According to the above definition, the contract associated to P is

$$\begin{array}{rcl} \mathsf{C}_P & = & \mathtt{Login}\,;\mathsf{C}_{(\overline{\mathtt{InvalidLogin}}\,;\,\mathtt{Login})^*\,;\,\overline{\mathtt{ValidLogin}}\,;\,Q} \\ \mathsf{C}_{(\overline{\mathtt{InvalidLogin}}\,;\,\mathtt{Login})^*\,;\,\overline{\mathtt{ValidLogin}}\,;\,Q} & = & \overline{\mathtt{InvalidLogin}}\,;\mathsf{C}_P \oplus \overline{\mathtt{ValidLogin}}\,;\mathsf{C}_Q \\ \mathsf{C}_Q & = & \cdots \end{array}$$

Observe that the contract C_1 in Example 3.1, which corresponds to the same activity P, is syntactically different from the one we obtain above. Using the techniques we develop in the next section, it is possible to demonstrate that the two contracts are equivalent. ■

A relevant property of C_P is an immediate consequence of Lemma 2.1.

Lemma 3.1. *For every P, the set $\mathsf{cnames}(\mathsf{C}_P)$ is finite.*

Given a BPEL abstract activity P, the contract C_P is compliance equivalent to it:

Theorem 3.1. $P \eqsim \mathsf{C}_P$.

Remark 3.1. *The Web service conversation language* WSCL *[BBB+02] describes* conversations *between two parties by means of an activity diagram (Figure 4). The diagram is made of* interactions *connected with each other by* transitions. *An interaction is a basic one-way or two-way communication between the client and the server. Two-way communications are just a shorthand for two sequential one-way interactions. Each interaction has a* name *and a list of* document types *that can be exchanged during its execution. A transition connects a* source *interaction with a* destination *interaction. A transition may be* labeled *by a document type if it is active only when a message of that specific document type was exchanged during the previous interaction.*

The diagram in Figure 4 describes the conversation of a service requiring clients to login before they can issue a query. After the query, the service returns a catalog. From this point on, the client can decide whether to purchase an item from the catalog or to logout and leave. In case of purchase, the service may either report that the purchase is successful, or that the item is out-of-stock, or that client's payment is refused. By interpreting names as message types, this e-commerce service can be described by the tuple:

$$\begin{array}{lcl} (\;\mathsf{C}_1 & = & \mathtt{Login}\,;(\overline{\mathtt{InvalidLogin}}\,;\mathsf{C}_1 \oplus \overline{\mathtt{ValidLogin}}\,;\mathsf{C}_2)\;, \\ \;\;\mathsf{C}_2 & = & \mathtt{Query}\,;\overline{\mathtt{Catalog}}\,;(\mathsf{C}_2 + \mathsf{C}_3 + \mathsf{C}_4)\;, \\ \;\;\mathsf{C}_3 & = & \mathtt{Purchase}\,;(\,\overline{\mathtt{Accepted}} \\ & & \qquad\quad \oplus\,\overline{\mathtt{InvalidPayment}}\,;(\mathsf{C}_3 + \mathsf{C}_4) \\ & & \qquad\quad \oplus\,\overline{\mathtt{OutOfStock}}\,;(\mathsf{C}_2 + \mathsf{C}_4)\,)\;, \\ \;\;\mathsf{C}_4 & = & \mathtt{Logout}\;, \\ \;\;\mathsf{C}_1 &) & \end{array}$$

There is a strict correspondence between unlabeled (respectively, labeled) transitions in Figure 4 and external (respectively, internal) choices in the contract. Recursion is used for modeling the cycles in the figure, namely the behaviors that can be iterated.

Theorem 3.1 allows us to define `flow` & `link`*-free* $\sqsubseteq$*-normal forms of abstract* BPEL *activities. Such* *normal forms are as intelligible as* WSCL *conversation diagrams, independently defined at Hewlett-Packard with the exact purpose of specifying the abstract interfaces supported by a concrete services.* ■

4. Coinductive compliance and must-testing

It is difficult to understand the general properties of the compliance preorder solely looking at Definition 2.2, because of the universal quantification over all (client) activities T and sequences of reductions. This makes direct proofs particularly challenging. For this reason, it is convenient to provide alternative characterizations of the compliance preorder that are supported by more manageable proof techniques. Among such characterizations, the so-called coinductive semantics are particularly appropriate, because proofs need to consider only single reduction steps instead of (infinite sets of) traces [Mil89]. In this section, we study a coinductive characterization of the compliance preorder and prove its correspondence with the semantics in Definition 2.2. Using this new definition of compliance we then demonstrate that the compliance preorder does coincide with a well-known semantics in concurrency theory: the must-testing preorder [Hen88, DH84]. This coincidence allows us to reuse the well-established theory developed for must-testing to reason about BPEL activities.

We follow the same conventions of Section 3 and let X and Y range over BPEL abstract activities *and* contracts without distinction.

Definition 4.1. *A* coinductive compliance *is a relation* $\mathcal{R}$ *such that* $\mathsf{X} \mathrel{\mathcal{R}} \mathsf{Y}$ *and* $\mathsf{X}\!\downarrow$ *implies*

1. $\mathsf{Y}\!\downarrow$*, and*
2. $\mathsf{Y} \Downarrow \mathrm{R}$ *implies* $\mathsf{X} \Downarrow \mathrm{S}$ *for some* $\mathrm{S} \subseteq \mathrm{R}$*, and*
3. $\mathsf{Y} \stackrel{\alpha}{\Longrightarrow}$ *implies* $\mathsf{X} \stackrel{\alpha}{\Longrightarrow}$ *and* $\mathsf{X}(\alpha) \mathrel{\mathcal{R}} \mathsf{Y}(\alpha)$.

We write $\preceq$ *for the largest coinductive compliance relation.*

According to this definition, a term X such that $\mathsf{X}\!\uparrow$ is the smallest one. When $\mathsf{X}\!\downarrow$, condition 1 requires the larger term Y to converge as well, since clients might rely on the convergence of X to complete successfully. Condition 2 states that each ready set R of Y (that is, each state reachable from Y by means of invisible moves only) is matched by a corresponding ready set S of X such that $\mathrm{S} \subseteq \mathrm{R}$. This is to say that Y exhibits a more deterministic behavior than X and that Y exposes *at least* the same capabilities as X. Condition 3 demands that Y should provide no more actions than those provided by X and that the corresponding continuations for any such action α be related by coinductive compliance. The rationale for using the continuations $\mathsf{X}(\alpha)$ and $\mathsf{Y}(\alpha)$ rather than simply any pair of derivatives of X and Y (as would be in a standard simulation relation) is motivated by the fact that clients are unaware of the internal choices performed by services. So, for example, $a\,;b + a\,;c \eqsim a\,;(b \oplus c)$ because, after interacting on a, a client of the service on the left hand side of $\eqsim$ is not aware of which state the service is in (it can be either b or c). By considering the continuations after a, we end up verifying $b \oplus c \mathrel{\mathcal{R}} b \oplus c$, which trivially holds for every coinductive compliance relation.

By now we have defined a range of compliance relations: one based the successful client-service interactions (Definition 2.2) and a coinductive one $\preceq$. Definition 2.2 can also be adapted according to the set of tests that we take into account. In particular, let $\sqsubseteq_C$ be the compliance relation when tests T are contracts and let $\sqsubseteq_{A+C}$ be the compliance relation of Definition 2.2 when tests T can be either abstract activities or contracts. Clearly $\mathsf{X} \sqsubseteq_{A+C} \mathsf{Y}$ implies both $\mathsf{X} \sqsubseteq \mathsf{Y}$ and $\mathsf{X} \sqsubseteq_C \mathsf{Y}$, while in principle the converse may be false. The following theorem guarantees the coincidence of all the compliance relations defined thus far and shows that $\preceq$ is a coinductive characterization of them.

Theorem 4.1. *For every* X *and* Y*, the following statements are equivalent:*

1. $\mathsf{X} \preceq \mathsf{Y}$*;*
2. $\mathsf{X} \sqsubseteq \mathsf{Y}$*;*
3. $\mathsf{X} \sqsubseteq_C \mathsf{Y}$*;*
4. $\mathsf{X} \sqsubseteq_{A+C} \mathsf{Y}$.

By relating a testing semantics and a coinductive semantics, Theorem 4.1 bridges the gap between the two techniques and allows one to choose the corresponding arguments interchangeably. Similar results have been provided for the lazy lambda calculus by Abramsky [Abr90], for the lambda calculus with local store by Pitts and Stark [PS93], and for process calculi by Boreale and Sangiorgi [BS98] and by Fournet and Laneve [FL01]. Thanks to Theorem 4.1, in the rest of the paper we will just use the symbol $\sqsubseteq\!\!\!\sim$ to denote both $\sqsubseteq\!\!\!\sim_{A+C}$ and $\sqsubseteq\!\!\!\sim_{C}$.

An application of Theorem 4.1 is to relate two apparently different testing semantics for abstract activities (and contracts): the compliance preorder and the *must-testing preorder* [Hen88]. To this aim, we recall the definition of the must preorder. In accordance with Definition 2.2, we let T to range over activities/contracts that may contain the special name e.

Definition 4.2 (Must preorder [DH87]). *A sequence of transitions* $\mathsf{X}_0 \,|_{\mathsf{N}}\, \mathsf{T}_0 \xrightarrow{\varepsilon} \mathsf{X}_1 \,|_{\mathsf{N}}\, \mathsf{T}_1 \xrightarrow{\varepsilon} \cdots$ *is a* maximal computation *if either it is infinite or the last term* $\mathsf{X}_n \,|_{\mathsf{N}}\, \mathsf{T}_n$ *is such that* $\mathsf{X}_n \,|_{\mathsf{N}}\, \mathsf{T}_n \not\longrightarrow$.

Let $\mathsf{X}\ \mathtt{must}\ \mathsf{T}$ *if, for every maximal computation* $\mathsf{X} \,|_{\mathsf{N}}\, \mathsf{T} = \mathsf{X}_0 \,|_{\mathsf{N}}\, \mathsf{T}_0 \xrightarrow{\varepsilon} \mathsf{X}_1 \,|_{\mathsf{N}}\, \mathsf{T}_1 \xrightarrow{\varepsilon} \cdots$, *there exists* $n \geq 0$ *such that* $\mathsf{T}_n \xrightarrow{\mathsf{e}}$.

We write $\mathsf{X} \sqsubseteq_{\mathtt{must}} \mathsf{Y}$ *if and only if, for every* T, $\mathsf{X}\ \mathtt{must}\ \mathsf{T}$ *implies* $\mathsf{Y}\ \mathtt{must}\ \mathsf{T}$.

Before showing the precise relationship between $\sqsubseteq\!\!\!\sim$ and $\sqsubseteq_{\mathtt{must}}$, let us comment on the differences between $\mathsf{X} \dashv \mathsf{T}$ and $\mathsf{X}\ \mathtt{must}\ \mathsf{T}$. The $\mathtt{must}$ relation is such that $\sigma\ \mathtt{must}\ \mathsf{e} + \rho$ holds for every σ, so that the observers of the form $\mathsf{e} + \rho$ are useless for discriminating between different (service) behaviors in $\sqsubseteq_{\mathtt{must}}$. However this is not the case for $\dashv$. For example $\mathsf{e} + a \not\dashv \overline{a}$ whilst $\overline{a}\ \mathtt{must}\ \mathsf{e} + a$. In our setting it makes no sense to declare that $\mathsf{e} + a$ is compliant with $\overline{a}$ with the justification that, at some point in a computation starting from $\mathsf{e} + a \mid \overline{a}$, the client can emit e. When a client and a service interact, actions cannot be undone. On the other hand we have $\mathsf{e} \oplus \mathsf{e} \dashv \Omega$ and $\Omega\ \not{\mathtt{must}}\ \mathsf{e} \oplus \mathsf{e}$. That is a (client) behavior compliant with a divergent (service) behavior is such that it is compliant with every (service) behavior. Hence $\mathsf{e} \oplus \mathsf{e}$ is useless for discriminating between different services. Historically, $\Omega\ \not{\mathtt{must}}\ \mathsf{e} \oplus \mathsf{e}$ has been motivated by the fact that the divergent process may prevent the observer from performing the one internal reduction that leads to success. In a distributed setting this motivation is no longer sustainable, since client and service will usually run independently on different processors. Finally, consider a divergent (client) behavior ρ. In the $\mathtt{must}$ relation such observer never succeeds unless $\rho \xrightarrow{\mathsf{e}}$. In the $\dashv$ relation such observer is compliant so long as all of its finite computations lead to a successful state. So, for example, the client behaviors $\mathsf{C} = a \,;\, \mathsf{e} \oplus \mathsf{C}$ and $a \,;\, \mathsf{e}$ have the same discriminating power as far as $\sqsubseteq\!\!\!\sim$ is concerned.

Notwithstanding the above different testing capabilities, $\sqsubseteq_{\mathtt{must}}$ and $\sqsubseteq\!\!\!\sim$ do coincide. As by Theorem 4.1, this is proved by demonstrating the equality of $\sqsubseteq_{\mathtt{must}}$ and $\preceq$.

Theorem 4.2. $\mathsf{X} \sqsubseteq_{\mathtt{must}} \mathsf{Y}$ *if and only if* $\mathsf{X} \sqsubseteq\!\!\!\sim \mathsf{Y}$.

Incidentally, Theorem 4.2, by relating $\sqsubseteq_{\mathtt{must}}$ and $\sqsubseteq\!\!\!\sim$, provides a coinductive characterization of $\sqsubseteq_{\mathtt{must}}$, which is, to the best of our knowledge, original in [LP07].

5. The Subcontract Relation

Theorems 3.1 and 4.2 show that $\sqsubseteq\!\!\!\sim$ and must-testing are appropriate relations to reason about (abstract) BPEL activities and their own contracts, since they are defined taking Web service clients as tests for discriminating between behaviors. Yet, there are contexts in which these relations are too strong, in particular when querying a repository of Web service contracts. In these cases, it is reasonable to work with a weaker notion of "service compatibility" that enables two useful properties called *width* and *depth extension*.

To illustrate, consider a service whose contract is $a \,;\, \overline{c}$, namely a service that receives a request a to which it answers with a response $\overline{c}$. It is reasonable to expect that, if the service is extended with a new functionality, let us say $a \,;\, \overline{c} + b \,;\, \overline{d}$, the clients of the original service will still comply with the extended one. Regrettably, this is not the case; for example, we have $\overline{a} \,;\, c \,;\, \mathsf{e} + \overline{b} \dashv a \,;\, \overline{c}$ and $\overline{a} \,;\, c \,;\, \mathsf{e} + \overline{b} \not\dashv a \,;\, \overline{c} + b \,;\, \overline{c}$, that is $\overline{a} \,;\, c \,;\, \mathsf{e} + \overline{b}$ succeeds with the original service, but fails with the extended one, witnessing that $a \,;\, \overline{c} \not\sqsubseteq\!\!\!\sim a \,;\, \overline{c} + b \,;\, \overline{c}$. This is an instance of width extension failure, whereby it is not possible to extend the behavior of a service with new operations offered by means of external choices. Similarly, extending the service $a \,;\, \overline{c}$ to $a \,;\, \overline{c} \,;\, b \,;\, \overline{d}$ is not allowed by $\sqsubseteq\!\!\!\sim$ because $\overline{a} \,;\, c \,;\, (\mathsf{e} + \overline{b}) \dashv a \,;\, \overline{c}$ and $\overline{a} \,;\, c \,;\, (\mathsf{e} + \overline{b}) \not\dashv a \,;\, \overline{c} \,;\, b \,;\, \overline{d}$. This is an instance of depth extension failure, whereby it is not possible to prolong the behavior of a service beyond its terminal states.

Both width and depth extension failures are a consequence of the fact that, among the clients of the original service (contract) $a\,;\overline{c}$, we respectively admit $\overline{a}\,; c\,; \mathsf{e}+\overline{b}$ and $\overline{a}\,; c\,; (\mathsf{e}+\overline{b})$ which are specifically (and possibly maliciously) crafted to *sense* an operation b not provided by the original service and to *fail* as soon as this operation is provided by the extended one. The existence of these clients is what makes compliance conservative, because $\sqsubseteq$ quantifies over all possible clients, including malicious ones. To define a coarser relation between contracts, one that allows both width and depth extensions, we restrict the set of clients (hence, of tests) that are compliant to (the contract of) an activity to those that never perform unavailable operations. To do so, following [LP07], we switch to more informative contracts than those described in Section 3. In particular, we consider pairs $\mathrm{I} : \sigma$, called *extended contracts*, where σ is a term as in Section 3 and $\mathrm{I} \supseteq \mathsf{actions}(\sigma)$ is a finite set of actions that defines the *interface* of the service whose behavior is described by σ. Then, we define a *subcontract relation* along the lines of Definition 2.2, except that we consider as tests only those clients that respect the interface of a contract, namely that do not request operations other than those in the interface of the contract.

Definition 5.1 (Subcontract relation)**.** *Let* $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau$ *if* $\mathrm{I} \subseteq \mathrm{J}$ *and, for every* $\mathrm{K} : \rho$ *such that* $\mathrm{K} \setminus \{\mathsf{e}\} \subseteq \overline{\mathrm{I}}$ *and* $\rho \dashv \sigma$ *implies* $\rho \dashv \tau$. *Let* $\approx$ *be* $\lesssim \cap \gtrsim$.

Notice that $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau$ only if $\mathrm{I} \subseteq \mathrm{J}$. This apparently natural prerequisite has substantial consequences on the properties of $\lesssim$ because it ultimately enables width and depth extensions, which are not possible in the $\sqsubseteq$ preorder. For instance, we have $\{a\} : a \lesssim \{a, b\} : a + b$ whilst $a \not\sqsubseteq a + b$ (width extension). Similarly we have $\{a\} : a \lesssim \{a, b\} : a\,; b$ whilst $a \not\sqsubseteq a\,; b$ (depth extension).

To highlight the relevant properties of $\lesssim$ and conforming to the same pattern used for $\sqsubseteq$, we provide an alternative characterization of $\lesssim$, which is also convenient in proofs. The characterization is similar to the one of Definition 4.1.

Definition 5.2. *A* coinductive subcontract *is a relation* $\mathcal{R}$ *such that if* $\mathrm{I} : \sigma \;\mathcal{R}\; \mathrm{J} : \tau$, *then* $\mathrm{I} \subseteq \mathrm{J}$ *and whenever* $\sigma\!\downarrow$ *we have:*

1. $\tau\!\downarrow$, *and*
2. $\tau \Downarrow \mathrm{R}$ *implies* $\sigma \Downarrow \mathrm{S}$ *and* $\mathrm{S} \subseteq \mathrm{R}$, *and*
3. $\alpha \in \mathrm{I}$ *and* $\tau \stackrel{\alpha}{\Longrightarrow}$ *imply* $\sigma \stackrel{\alpha}{\Longrightarrow}$ *and* $\mathrm{I} : \sigma(\alpha) \;\mathcal{R}\; \mathrm{J} : \tau(\alpha)$.

Definition 5.2 is structurally very similar to Definition 4.1, with two relevant differences: the first one is the condition $\mathrm{I} \subseteq \mathrm{J}$, which follows directly from Definition 5.1; the second and fundamental one is that, in condition (3), only the actions α that were already present in the smaller contract are taken into account when considering the continuations. This way, any behavior provided by the larger contract that follows an action α which is not in the interface of the smaller contract need is ignored. Definition 5.2 completely characterizes the subcontract relation:

Theorem 5.1. $\lesssim$ *is the largest coinductive subcontract relation.*

The next proposition summarizes the most relevant properties of $\lesssim$ in a formal way. In particular, it emphasizes the width and depth extensions allowed by $\lesssim$ but forbidden in $\sqsubseteq$ and in the must-testing preorder. The proof of these properties is easy using the alternative characterization of $\lesssim$ in Definition 5.2).

Proposition 5.1. *The following properties hold:*

1. *If* $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau$ *and* $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau'$, *then* $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau \oplus \tau'$*;*
2. *if* $\mathrm{I} : \mathsf{0} \lesssim \mathrm{J} : \tau$, *then* $\mathrm{I} : \sigma \lesssim \mathrm{J} : \sigma + \tau$ *(*width extension*);*
3. *if* $\mathrm{I} : \mathsf{0} \lesssim \mathrm{J} : \tau$, *then* $\mathrm{I} : \sigma \lesssim \mathrm{J} : \sigma\{^{\tau}/_{\mathsf{0}}\}$, *where* $\sigma\{^{\tau}/_{\mathsf{0}}\}$ *is the replacement of every occurrence of the contract name* $\mathsf{0}$ *with* τ *(*depth extension*).*

Item 1 states that the clients that are compliant with both contracts $\mathrm{J} : \tau$ and $\mathrm{J} : \tau'$ are also compliant with services that internally decide to behave according to either τ or τ'. Item 2 gives sufficient conditions for width extensions of Web services: a Web service may be upgraded to offer additional functionalities without affecting the set of clients it satisfies, so long as the names of such new functionalities were not present in the original service. Here the premise $\mathrm{I} : \mathsf{0} \lesssim \mathrm{J} : \tau$ formalizes the concept of "new functionality": any action α such that $\tau \stackrel{\alpha}{\Longrightarrow}$ must be in $\mathrm{J} \setminus \mathrm{I}$ and in particular it cannot be an action of σ. Additionally, the same premise implies that $\tau{\downarrow}$, because we have $0{\downarrow}$ (see Definition 5.2). Item 3 is similar to item 2, but concerns depth extensions, that is the ability to extend the conversation offered by a service, provided that the additional conversation begins with new functionalities not present in the original service. In fact, item 2 can be seen as a special case of item 3, if we consider the contract $\mathrm{I} : \sigma + 0$ instead of simply $\mathrm{I} : \sigma$.

The precise relationship between $\lesssim$ and $\sqsubseteq$ is expressed by the following statement.

Proposition 5.2. $\mathrm{I} : \sigma \approx \mathrm{J} : \tau$ *if and only if* $\sigma \eqsim \tau$ *and* $\mathrm{I} = \mathrm{J}$.

6. Duality

We now analyze the problem of querying a repository of BPEL activities, where every activity P is modeled by the extended contract $\mathrm{I}_P : \mathsf{C}_P$ such that $\mathrm{I}_P = \mathsf{actions}(\mathsf{C}_P)$ and C_P is defined in Section 3. The basic problem for querying such a repository is that, given a client's extended contract $\mathrm{K} : \rho$, one wishes to find all the pairs $\mathrm{I} : \sigma$ such that $\mathrm{K} \setminus \{\mathsf{e}\} \subseteq \overline{\mathrm{I}}$ and $\rho \dashv \sigma$.

We attack this problem in two steps: first of all, we compute *one particular extended contract* $\overline{\mathrm{K}} \setminus \{\overline{\mathsf{e}}\} : \mathsf{D}^{\mathrm{K}}_{\rho}$, called *dual* of $\mathrm{K} : \rho$, such that $\rho \dashv \mathsf{D}^{\mathrm{K}}_{\rho}$; second, we collect all the services in the registry whose extended contract is larger (according to $\lesssim$) than this one. To be sure that no suitable service is missing in the answer to the query, the *dual* of a client $\mathrm{K} : \rho$ should be a pair $\overline{\mathrm{K}} \setminus \{\overline{\mathsf{e}}\} : \mathsf{D}^{\mathrm{K}}_{\rho}$ that it is the *smallest one* (according to $\lesssim$) that satisfies the client $\mathrm{K} : \rho$. We call such pair the *principal dual extended contract* of $\mathrm{K} : \rho$.

In defining the principal dual extended contract, it is convenient to restrict the definition to those client's behaviors ρ that never lead to 0 without emitting e. For example, the behavior $a \,;\, \mathsf{e} + b$ describes a client that succeeds if the service proposes $\overline{a}$, but that fails if the service proposes $\overline{b}$. As far as querying is concerned, such behavior is completely equivalent to $a \,;\, \mathsf{e}$. As another example, the degenerate client behavior 0 is such that no service will ever satisfy it. In general, if a client is unable to handle a particular action, like b in the first example, it should simply omit that action from its behavior. We say that a (client) extended contract $\mathrm{K} : \rho$ is *canonical* if, whenever $\rho \stackrel{\varphi}{\Longrightarrow} \rho'$ is maximal, then $\varphi = \varphi' \mathsf{e}$ and e does not occur in φ'. For example $\{a, \mathsf{e}\} : a \,;\, \mathsf{e}$, $\{a\} : \mathsf{C}$, where $\mathsf{C} = a \,;\, \mathsf{C}$, and $\varnothing : \Omega$ are canonical; $\{a, b, \mathsf{e}\} : a \,;\, \mathsf{e} + b$ and $\{a\} : \mathsf{C}'$, where $\mathsf{C}' = a \oplus \mathsf{C}'$, are not canonical.

Observe that Lemma 2.1 also applies to contracts. Therefore it is possible to extend the notions in Definition 2.1, by replacing activities with contracts.

Definition 6.1 (Dual contract). *Let* $\mathrm{K} : \rho$ *be a canonical extended contract. The* dual *of* $\mathrm{K} : \rho$ *is* $\overline{\mathrm{K}} \setminus \{\overline{\mathsf{e}}\} : \mathsf{D}^{\mathrm{K}}_{\rho}$ *where* $\mathsf{D}^{\mathrm{K}}_{\rho}$ *is the contract name defined as follows:*

$$\mathsf{D}^{\mathrm{K}}_{\rho} \stackrel{\text{def}}{=} \begin{cases} \Omega & \text{if } \mathsf{init}(\rho) = \{\mathsf{e}\} \\ \sum_{\substack{\rho \Downarrow \mathrm{R} \\ \mathrm{R}\setminus\{\mathsf{e}\} \neq \varnothing}} \Big(\underbrace{0 \oplus}_{\text{if } \mathsf{e} \in \mathrm{R}} \bigoplus_{\alpha \in \mathrm{R}\setminus\{\mathsf{e}\}} \overline{\alpha} \,;\, \mathsf{D}^{\mathrm{K}}_{\rho(\alpha)} \Big) + \mathsf{E}^{\mathrm{K}\setminus\mathsf{init}(\rho)} & \text{otherwise} \end{cases}$$

$$\mathsf{E}^{\mathrm{S}} \stackrel{\text{def}}{=} \underbrace{0 \oplus \bigoplus_{\alpha \in \mathrm{S}} \overline{\alpha} \,;\, \Omega}_{\text{if } \mathrm{S} \neq \varnothing}$$

Few comments about $\mathsf{D}^{\mathrm{K}}_{\rho}$, when $\mathsf{init}(\rho) \neq \{\mathsf{e}\}$, follow. In this case, the behavior ρ may autonomously transit to different states, each one offering a particular ready set. Thus the dual behavior leaves the choice to the client: this is the reason for the external choice in the second line. Once the state has been chosen, the client offers to the service a spectrum of possible actions: this is the reason for the internal choice underneath the sum $\sum$.

The contract $\mathsf{E}^{\mathrm{K}\setminus\mathsf{init}(\rho)}$ covers all the cases of actions that are allowed by the interface and that are not offered by the client. The point is that the dual operator must compute the principal (read, the smallest) service contract that satisfies the client, and the smallest convergent behavior with respect to a nonempty (finite) interface S is $0 \oplus \bigoplus_{\alpha \in \mathrm{S}} \overline{\alpha} \,;\, \Omega$. The 0 summand accounts for the possibility that none of the actions in $\mathrm{K} \setminus \mathsf{init}(\rho)$ is present. The external choice "+" distributes the proper dual contract over the internal choice of all the actions in $\mathrm{K} \setminus \mathsf{init}(\rho)$. For example, $\mathsf{D}^{\{a,\overline{a},\mathsf{e}\}}_{a \,;\, \mathsf{e}} = \overline{a} \,;\, \Omega + (0 \oplus a \,;\, \Omega)$. The dual of a divergent (canonical) client $\{a, \mathsf{e}\} : \mathsf{C}$, where $\mathsf{C} = a\,;\mathsf{e} \oplus \mathsf{C}$, is also well defined: $\mathsf{D}_{\mathsf{C}}^{\{a,\mathsf{e}\}} = \overline{a}\,;\Omega$. We finally observe that the definition also accounts for duals of non-terminating clients, such as $\{a\} : \mathsf{C}'$, where $\mathsf{C}' = a\,;\mathsf{C}'$. In this case, $\mathsf{D}_{\mathsf{C}'}^{\{a\}} = \overline{a}\,;\mathsf{D}_{\mathsf{C}'}^{\{a\}}$.

Similarly to the definition of contract names C_P in Section 3, it is possible to prove that $\mathsf{D}_\rho^{\mathrm{K}}$ is well defined.

Lemma 6.1. *For every* $\mathrm{K} : \rho$*, the set* $\mathsf{cnames}(\mathsf{D}_\rho^{\mathrm{K}})$ *is finite.*

Example 6.1. *We illustrate the definition of dual of an extended contract on a potential client of the service in the Example 3.1. This simple client* `Login`*s and, when the credentials have been accepted, performs exactly one* `Query` *to the* `Catalog` *and then* `Logout`*s. The contract of such a client is defined by the following equations:*

$$\begin{array}{lcl}\mathsf{C}'_1 & = & \overline{\texttt{Login}}\,;\mathsf{C}'_2\\ \mathsf{C}'_2 & = & \texttt{InvalidLogin}\,;\mathsf{C}'_1 + \texttt{ValidLogin}\,;\mathsf{C}'_3\\ \mathsf{C}'_3 & = & \overline{\texttt{Query}}\,;\mathsf{C}'_4\\ \mathsf{C}'_4 & = & \texttt{Catalog}\,;\mathsf{C}'_5\\ \mathsf{C}'_5 & = & \overline{\texttt{Logout}}\,;\mathsf{e}\end{array}$$

Let $\mathrm{K} = \{\overline{\texttt{Login}}, \texttt{InvalidLogin}, \texttt{ValidLogin}, \overline{\texttt{Query}}, \texttt{Catalog}, \overline{\texttt{Logout}}, \mathsf{e}\}$ *and notice that* $\mathrm{K} : \mathsf{C}'_1$ *is canonical. Its principal dual extended contract is* $\overline{\mathrm{K}} \setminus \{\overline{\mathsf{e}}\} : \mathsf{D}_{\mathsf{C}'_1}^{\mathrm{K}}$*, where*

$$\begin{array}{lcl}\mathsf{D}_{\mathsf{C}'_1}^{\mathrm{K}} & = & \texttt{Login}\,;\mathsf{D}_{\mathsf{C}'_2}^{\mathrm{K}} + \mathsf{E}^{\mathrm{K}\setminus\{\overline{\texttt{Login}}\}}\\ \mathsf{D}_{\mathsf{C}'_2}^{\mathrm{K}} & = & (\overline{\texttt{InvalidLogin}}\,;\mathsf{D}_{\mathsf{C}'_1}^{\mathrm{K}} \oplus \overline{\texttt{ValidLogin}}\,;\mathsf{D}_{\mathsf{C}'_3}^{\mathrm{K}}) + \mathsf{E}^{\mathrm{K}\setminus\{\texttt{InvalidLogin},\texttt{ValidLogin}\}}\\ \mathsf{D}_{\mathsf{C}'_3}^{\mathrm{K}} & = & \texttt{Query}\,;\mathsf{D}_{\mathsf{C}'_4}^{\mathrm{K}} + \mathsf{E}^{\mathrm{K}\setminus\{\overline{\texttt{Query}}\}}\\ \mathsf{D}_{\mathsf{C}'_4}^{\mathrm{K}} & = & \overline{\texttt{Catalog}}\,;\mathsf{D}_{\mathsf{C}'_5}^{\mathrm{K}} + \mathsf{E}^{\mathrm{K}\setminus\{\texttt{Catalog}\}}\\ \mathsf{D}_{\mathsf{C}'_5}^{\mathrm{K}} & = & \texttt{Logout}\,;\Omega + \mathsf{E}^{\mathrm{K}\setminus\{\overline{\texttt{Logout}}\}}\end{array}$$

Let $\mathrm{K}' = \overline{\mathrm{K}} \cup \{\texttt{Purchase}, \overline{\texttt{Accepted}}, \overline{\texttt{InvalidPayment}}, \overline{\texttt{OutOfStock}}\}$*. We invite the reader to verify that* $\overline{\mathrm{K}} \setminus \{\overline{\mathsf{e}}\} : \mathsf{D}_{\mathsf{C}'_1}^{\mathrm{K}} \lesssim \mathrm{K}' \setminus \{\overline{\mathsf{e}}\} : \mathsf{C}_1$*, where* C_1 *has been defined in Example 3.1.* □

A basic property of the dual contract of $\mathrm{K} : \rho$ is that it defines the behavior of the least service compliant with $\mathrm{K} : \rho$. This property, known in type theory as *principal type property*, guarantees that queries to service registries are answered with the largest possible set of compliant services.

Theorem 6.1. *Let* $\mathrm{K} : \rho$ *be a canonical extended contract. Then:*

1. $\rho \dashv \mathsf{D}_\rho^{\mathrm{K}}$*;*
2. *if* $\overline{\mathrm{K}} \setminus \{\overline{\mathsf{e}}\} \subseteq \mathrm{S}$ *and* $\rho \dashv \sigma$*, then* $\overline{\mathrm{K}} \setminus \{\overline{\mathsf{e}}\} : \mathsf{D}_\rho^{\mathrm{K}} \lesssim \mathrm{S} : \sigma$*.*

A final remark is about the computational complexity of the discovery algorithm. Deciding $\lesssim$ is EXPTIME-complete in the size of the contracts [AIS11], and this cost should, in principle, be multiplied by the number of services in the repository. However, since $\lesssim$ is (obviously) transitive (see Definition 5.1), it is reasonable to assume that Web service are ordered according to $\lesssim$ as soon as they are entered into the registry. Therefore, at runtime the $\lesssim$ relation must be decided only for the $\lesssim$-minimal services, the remaining ones being determined by the (pre-computed) transitive closure.

7. Choreographies

The theory of contracts developed so far is based on the interaction between one client and one service. The aim of this section is to study some properties of the subcontract relation in a broader context, considering systems composed of an arbitrary number of services. A *choreography* describes the (parallel) composition of n services (called participants) that communicate with each other by means of private names and with the external world by means of public names. Standard languages for describing choreographies, such as the Web Service Choreography Description Language (WS-CDL [KBR+05]), allow an architect of a distributed system to describe the inter-participant interactions by giving a global description (choreography) of the system, rather than describing the behavior of each single participant (end-point behavior). In particular, the global description determines where and when a communication has to happen. That is, the architect decides that e.g. there will be a message from a participant A to a participant B and overlooks how this communication will be implemented. Eventually, the global description is *projected* into the local descriptions of its participants. In this section, following a standard approach in the literature, see for example [CHY07] and [BZ07], we identify the local description of a participant with its extended contract, and we represent a choreography as the parallel composition of the contracts of its participants. More formally, we represent a choreography as an *ordered* version of the n-ary flow and link term in Section 2:

$$\Gamma \quad ::= \quad \prod^{A}(\mathrm{I}_1 : \sigma_1, \dots, \mathrm{I}_n : \sigma_n)$$

where A is a subset of names representing the private names of the choreography. We write $\Gamma[i \mapsto \mathrm{J} : \rho]$ for the choreography that is the same as Γ except that (the extended contract of) the i-th participant has been replaced by $\mathrm{J} : \rho$.

The transition relation of choreographies is defined using that of behaviors by the following rules, where $\Gamma = \prod^{A}(\mathrm{I}_1 : \sigma_1, \dots, \mathrm{I}_n : \sigma_n)$:

$$\frac{\sigma \xrightarrow{\mu} \sigma' \qquad \mu \notin A \cup \overline{A}}{\Gamma[i \mapsto \mathrm{I} : \sigma] \xrightarrow{\mu} \Gamma[i \mapsto \mathrm{I} : \sigma']} \qquad \frac{i \neq j \qquad \sigma \xrightarrow{\alpha} \sigma' \qquad \tau \xrightarrow{\overline{\alpha}} \tau' \qquad \alpha \in A \cup \overline{A}}{\Gamma[i \mapsto \mathrm{I} : \sigma][j \mapsto \mathrm{J} : \tau] \xrightarrow{\varepsilon} \Gamma[i \mapsto \mathrm{I} : \sigma'][j \mapsto \mathrm{J} : \tau']}$$

That is, a choreography $\Gamma = \prod^{A}(\mathrm{I}_1 : \sigma_1, \dots, \mathrm{I}_n : \sigma_n)$ is akin to a (compound) service whose interface is $\mathsf{actions}(\Gamma) \stackrel{\text{def}}{=} \bigcup_{1 \le i \le n} \mathrm{I}_i \setminus (A \cup \overline{A})$ and whose behavior is the combination of the behaviors of the end-point projections running in parallel.

Having provided choreographies with a transition relation, the notions of *convergence*, *divergence*, and *ready set* can be immediately extended from Definition 2.1 to choreographies. Similarly, the notion of compliance may be extended in order to relate the behavior of a client with (the behavior of) a choreography, which we denote by $\rho \dashv \Gamma$. More precisely, we say that an extended (client) contract $\mathrm{K} : \rho$ *is compliant with* the choreography Γ if $\overline{\mathrm{K}} \setminus \{\mathsf{e}\} \subseteq \mathsf{actions}(\Gamma)$ and $\rho \dashv \Gamma$, as in Definition 2.2.

In the remaining part of the section we show that the subcontract relation (Definition 5.1) suitably addresses the problem of contract refinement, namely it allows one to replace a given choreography Γ with a refined one Γ', where some or all the participants behave according to refined contracts, still preserving the correctness of the overall system. By correctness we mean that every client that was compliant with the original choreography is still compliant with the refined one. Technically, this shows that under mild conditions the relation $\lesssim$ is a pre-congruence with respect to parallel composition of services, and therefore can be used for the modular refinement of complex systems.

Definition 7.1 (Choreography refinement)**.** *Let* $\Gamma = \prod^{A}(\mathrm{I}_1 : \sigma_1, \dots, \mathrm{I}_n : \sigma_n)$ *and* $\Gamma' = \prod^{A}(\mathrm{J}_1 : \tau_1, \dots, \mathrm{J}_n : \tau_n)$ *be choreographies. We say that* Γ' *is a* refinement *of* Γ *if:*

1. $\mathrm{I}_i : \sigma_i \lesssim \mathrm{J}_i : \tau_i$ *for every* $1 \le i \le n$*;*
2. $(\mathrm{J}_i \setminus \mathrm{I}_i) \cap \mathrm{I}_j = \varnothing$ *for every* $1 \le i, j \le n$*.*

Refinement defines a "safe" replacement of activities in a choreography with refined ones (such as their implementations). The replacing activities may have more capabilities than those offered by the replaced ones (condition (1)), although the set A of private names must be the same in both the original and the refined choreography. This is to make sure that the original choreography specification is respected in the refinement. Additionally, there must be no interferences between the additional capabilities of the refined choreography with respect to those in the original choreography (condition (2)). In particular, every action that is introduced in the refinement of a peer must be disjoint from any other action of the original choreography. To illustrate the relevance of condition (2), consider the choreographies

$$\Gamma_1 \stackrel{\text{def}}{=} \prod^{\varnothing}(\{a\} : a, \{b, \overline{c}\} : b\,;\overline{c}) \qquad \text{and} \qquad \Gamma_2 \stackrel{\text{def}}{=} \prod^{\varnothing}(\{a, b\} : a + b, \{b, \overline{c}\} : b\,;\overline{c})$$

and observe that each extended contract in Γ_1 is a subcontract of the corresponding one in Γ_2, that is Γ_1 and

Γ_2 satisfy condition (1) of Definition 7.1. Nonetheless, extending the first participant in Γ_1 to $\{a, b\} : a + b$ in Γ_2 introduces an interference on b, which is an operation provided also by the second participant. For instance, the client behavior $\overline{b}\,;c\,;\mathtt{e}$ is compliant with Γ_1 but not with Γ_2. Condition (2) prevents Γ_2 from being a refinement of Γ_1 by forbidding the introduction of these interferences. In practice, this condition is not restrictive since operation names usually include the name of the participant which provides them.

We now prove a soundness result for the notion of refinement. The result does not rely on any particular property (e.g. deadlock freedom) of the choreography itself. We merely show that, from the point of view of a client interacting with a choreography as a whole, the refinement of the choreography is unobservable.

Theorem 7.1. *Let* $\kappa : \rho$ *be compliant with* Γ_1 *and* Γ_2 *be a refinement of* Γ_1*. Then* $\kappa : \rho$ *is also compliant with* Γ_2*.*

8. Related Work

In this section, we relate our results to previous contributions in the literature. Since this paper covers several areas, we organize the analysis of related works in different paragraphs.

Semantics of BPEL. This paper defines a formal semantics of BPEL abstract activities, which has been the subject of several works (a comparative summary of the contributions in this area is given in [OVvdA+07]). In particular, the contributions in the literature define BPEL activities in terms of some finite model, such as Petri nets or finite state automata. Unlike these works, our approach focuses on the interactions between a BPEL process and the environment in which it executes. This is a well-known approach for the semantics of processes in concurrency theory (see, for instance, [Mil82, Hen88, Mil89]), but it is original for BPEL. In particular, it allows us to give a natural semantics of BPEL activities based on client-server interactions, which we call compliance preorder. However, the direct definition of compliance preorder (Definition 2.2) quantifies over both contexts and reductions; this makes direct proofs particularly difficult. To overcome this problem, following [Mil89] we define an alternative semantics formulated in a coinductive way – the coinductive compliance (Definition 4.1) – which supports a powerful proof technique. Our approach has also suggested the behavioral types for BPEL abstract activities: they are simply the normal forms of the compliance preorder, namely tauless CCS processes, a calculus developed by De Nicola and Hennessy in a number of contributions [Mil82, DH87, Hen88].

Behavioral types. Contracts are a form of behavioral type insofar they describe the behavior of a communicating process (in our case, a BPEL activity) in terms of the order of interaction events the process is supposed to perform. The literature on behavioral types is extensive. Among the pioneering works using CCS-like processes as types we mention [NN94, IK01, CRR02]. Perhaps the family of behavioral types that relates more closely to our contracts is session types [Hon93, HVK98]. Session types describe the order, direction and type of messages that are supposed to be exchanged over a communication channel and enable forms of static analysis to ensure the absence of communication errors, protocol fidelity, and limited forms of progress. Both contracts and session types distinguish between *internal choices*, those autonomously performed by an entity that behaves according to the type, and *external choices*, those determined by the environment in which the entity executes. These analogies carry on to some extent at the semantic level, since subtyping relations for session types are known to support width extensions [GH05]. Contracts differ from session types, and from the other kinds of behavioral types mentioned above, at various levels. A major distinction is that we use contracts for providing abstract descriptions of *whole* process behaviors, whereas other behavioral types, session types in particular, describe the behavior of a process with respect to a single communication channel. In our case, the actions occurring in a contract can identify operations, methods, messages, signals, and invocations/interactions are not required to occur within the scope of a single communication channel. For example, in Section 7 we use actions for describing the interactions of a choreography in which we assume that no interference is possible in communications between different pairs of participants. That is to say that we assume peer-to-peer, independent communications between pairs of participants. Another difference is the granularity at which interactions are described. In our contract language actions are *atomic* and have no structure. Conversely, session types usually allow fine-grained descriptions of the content of messages and even forms of higher-order communications. Our contract language can be extended for supporting forms of channel mobility, as described for example in [CP09]. We have already mentioned that session type theories are usually equipped with a subtyping relation [GH05] that shares common traits with the subcontract relation (Definition 5.1 and Proposition 5.1). With respect to [GH05] our contract language can express more general forms of interaction. In particular, subtyping for session types is often restricted in such a way that only choices of the same kind can be related by subtyping, meaning that internal and external choices can never be related. For example, the relation $\{a, b\} : a \oplus b \preceq \{a, b\} : a + b$ does *not* hold in session type theories. Also, depth extension of session types is generally unsupported since session interactions are meant to guarantee progress for all participants of a session. Asymmetric, session-based interactions have been considered in [BCd09]. A thorough analysis of the relationship between contracts and session types and between subcontract and subtyping relations have been investigated in [Ber13, BH12, LP08].

Types as search keys. The idea of using abstract description of software entities – types in particular – as keys for querying libraries/repositories is not new and can be traced back to [Rit93, Cos95]. The cornerstone of these approaches is the definition of a notion of type equivalence, therein called *type isomorphism*, that is used for comparing the type/key of the entities in a repository with the type/key of the ideal entity that is required in a particular context. The precise notion of type equivalence must be carefully crafted to meet three contrasting criteria: it should be sufficiently coarse to maximize the number of results of a query; it should be sufficiently narrow so that only compatible entities are identified; finally, it should be efficient to decide, given the potentially large size of libraries/repositories. All the aforementioned works refer to a sequential setting where types describe the input/output behaviors of functions. Our work, which started off with [CCLP06], shares similar objectives, but in the context of concurrent, usually distributed, Web service compositions whose type we call contract. The definition of contract equivalences based on a notion of client satisfaction – compliance – is a straightforward way of meeting the first criterion. In [CCLP06] the subcontract relation (over finite contracts) enjoys the width extension property illustrated in Section 4 but it lacks transitivity. Transitivity, while not being strictly necessary as far as querying and searching are concerned, is a key ingredient for meeting the third criterion. Indeed, if the subcontract relation is transitive, then databases of Web service contracts can be organized in accordance with the subcontract relation, so as to reduce the run time spent for executing queries: only services with a minimal contract must be checked for answering a query, the others are implicitly determined by transitivity and can be precomputed when the repository is populated. The problem of defining a sufficiently coarse, transitive subcontract relation has also been addressed in [CGP09]. The authors of [CGP09] make the assumption that client and service can be mediated by a *filter*, which prevents potentially dangerous interactions by dynamically changing the interface of the service as it is seen by the client. Even more expressive filters, akin to actual orchestrators, have been investigated in [Pad08, Pad09, Pad10]. In these cases the subcontract relation can be extended even further, by allowing (partial) permutation of actions whenever these do not disrupt the flow of messages between client and service. The present work, on the other hand, defines a transitive subcontract that supports both width and depth extensions of services without assuming any filter/orchestrator that mediates the interaction between the service and its clients. Clearly, the resulting subcontract relation is not as coarse as those defined in these works, but it embeds a safe substitution principle without implying any runtime overhead. In the present work we also consider divergence, which is not addressed in [CGP09, Pad08, Pad09, Pad10]. Using a framework similar to the one we have adopted in this work, subcontract and subtyping relations that preserve liveness properties have been defined in [BMPR09, Pad13, Pad14].

Choreographies. There has been a growing interest in studying the formal relationship between global description of interaction protocols – *choreographies* – and the local behavioral description of their participants. For example, in [CHY07, HYC08], Carbone *et al.* define a model of WS-CDL and relate this model to session types. The contract language that we have studied in this work is not aimed at providing choreographic descriptions. Rather, it is appropriate for describing the local behavior of choreography participants. In this sense, it closer in spirit to the contract languages studied by Bravetti *et al.* in a number of contributions [BLZ09, BZ09b, BZ08, BZ09a]. The main difference between these works and our own is the use of contract *interfaces* enabling forms of service discovery and replacement (width and depth extensions) that are normally unsound according to the well-known behavioral equivalences such as must- and should-testing.

XML **schemas.** Regarding schemas, which are currently part of BPEL contracts, it is worth mentioning that they have been the subject of formal investigation by several research projects [HP03, BCF03, CLP09]. This work aims at pursuing a similar objective, but moving from the description of data to the description of behaviors.

9. Conclusions

In this contribution we have studied a formal theory of Web service abstract (behavioral) definitions as normal forms of a natural semantics for BPEL activities. Our abstract definitions may be effectively used in any query-based system for service discovery because they support a notion of principal dual contract. This operation is currently done in an *ad hoc* fashion using search engines or similar technologies.

It should be noted that our framework rests on a correspondence between abstract activities as defined in Section 2 and BPEL activities, noted "BPEL Client", "BPEL Service 1", and "BPEL Service 2" in Figure 1, which cannot be completely formalized for the simple reason that BPEL is not equipped with a formal semantics. In addition, BPEL activities define details about internal computations that are omitted in the corresponding abstractions. In fact, models of BPEL services are infinite state, while abstract BPEL activities have finite models. However, our abstract BPEL activities allow us to over-approximate BPEL activities as far as the observable communication behavior is concerned by increasing non-determinism (for example, by representing a deterministic conditional construct as a non-deterministic choice $P \oplus Q$). Roughly speaking, this means for instance that $P_1 \lesssim$ "BPEL Service 1" in Figure 1 and we give some evidence of this fact in Section 2. A side-effect of this added non-determinism is that there may be BPEL clients that successfully complete their interaction with a BPEL service, while the compliance cannot be assessed between the corresponding abstract BPEL activities. In general, such approximations are widespread in (behavioral) type theories and are in fact one of the key ingredients that make them decidable.

Several future research directions stem from this work. On the technical side, a limit of our technique is that BPEL activities are "static", *i.e.* they cannot create other services on the fly. This constraint implies the finiteness of models and, for this reason, it is possible to effectively associate an abstract description to activities. However, this impacts on scalability, in particular when services adapt to peaks of requests by creating additional services. It is well-known that such an additional feature makes models to be infinite states and requires an approximate inferential process to extract abstract descriptions from activities. Said otherwise, extending our technique to full CCS or π-calculus amounts to defining abstract finite models such that Theorem 3.1 does not hold anymore. For this reason, under- and over-estimations for services and clients, respectively, must be provided.

Another interesting technical issue concerns the extension of our study to other semantics for BPEL activities, such as the preorder in [BZ09b], or even to weak bisimulation (which has a polynomial computational cost). To this aim, the axiomatizations that have been defined for these semantics might be used to select normal forms of processes and in turn to determine their contracts. However it is not clear whether such semantics admit a principal dual contract or not.

It is also interesting to prototyping our theory and experimenting it on some existing repository, such as `http://www.service-repository.com/`. To this aim we might re-use tools that have been already developed for the must testing, such as the concurrency workbench [CPS93].

References

[Abr90] Samson Abramsky. The lazy lambda calculus. In *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1990.

[AIS11] L. Aceto, Anna Ingolfsdottir, and Jiri Srba. The algoritmics of bisimilarity. In D. Sangiorgi and J. Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, volume 52 of *Cambridge Tracts in Theoretical Computer Science*, chapter 3, pages 100–172. Cambridge University Press, 2011.

[Alv07] Alexandre Alves et al. *Web Services Business Process Execution Language Version 2.0*, January 2007. `http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.html`.

[BBB+02] Arindam Banerji, Claudio Bartolini, Dorothea Beringer, Venkatesh Chopella, et al. *Web Services Conversation Language (*WSCL*) 1.0*, March 2002. `http://www.w3.org/TR/2002/NOTE-wscl10-20020314`.

[BCd09] Franco Barbanera, Sara Capecchi, and Ugo de'Liguoro. Typing asymmetric client-server interaction. In *Fundamentals of Software Engineering, Third IPM International Conference, FSEN 2009, Kish Island, Iran, April 15-17, 2009, Revised Selected Papers*, volume 5961 of *Lecture Notes in Computer Science*, pages 97–112. Springer, 2009.

[BCF03] Véronique Benzaken, Giuseppe Castagna, and Alain Frisch. CDuce: an XML-centric general-purpose language. *SIGPLAN Notices*, 38(9):51–63, 2003.

[Ber13] Giovanni Bernardi. *Behavioural Equivalences for Web Services.* PhD thesis, University of Dublin, 2013.

[BGZ09] Nadia Busi, Maurizio Gabbrielli, and Gianluigi Zavattaro. On the expressive power of recursion, replication and iteration in process calculi. *Mathematical Structures in Computer Science*, 19(6):1191–1222, 2009.

[BH12] Giovanni Bernardi and Matthew Hennessy. Modelling session types using contracts. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1941–1946, New York, NY, USA, 2012. ACM.

[BHR84] Stephen D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

[BKL01] D. Beringer, H. Kuno, and M. Lemon. *Using* WSCL *in a* UDDI *Registry 1.0*, 2001. UDDI Working Draft Best Practices Document, http://xml.coverpages.org/HP-UDDI-wscl-5-16-01.pdf.

[BLZ09] Mario Bravetti, Ivan Lanese, and Gianluigi Zavattaro. Contract-driven implementation of choreographies. In *Trustworthy Global Computing*, volume 5474 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2009.

[BMPR09] Michele Bugliesi, Damiano Macedonio, Luca Pino, and Sabina Rossi. Compliance preorders for web services. In *WS-FM*, volume 6194 of *Lecture Notes in Computer Science*, pages 76–91. Springer, 2009.

[BS98] M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. In *Logic in Computer Science, 1998. Proceedings. Thirteenth Annual IEEE Symposium on*, pages 165–175, Jun 1998.

[BZ07] Mario Bravetti and Gianluigi Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *Pre-proceedings of 6th Symposium on Software Composition*, 2007.

[BZ08] Mario Bravetti and Gianluigi Zavattaro. A foundational theory of contracts for multi-party service composition. *Fundam. Inform.*, 89(4):451–478, 2008.

[BZ09a] Mario Bravetti and Gianluigi Zavattaro. Contract-based discovery and composition of web services. In *SFM'09*, volume 5569 of *Lecture Notes in Computer Science*, pages 261–295. Springer, 2009.

[BZ09b] Mario Bravetti and Gianluigi Zavattaro. A theory of contracts for strong service compliance. *Mathematical Structures in Computer Science*, 19:601–638, 5 2009.

[CCLP06] S. Carpineti, G. Castagna, C. Laneve, and L. Padovani. A formal account of contracts for Web Services. In *WS-FM, 3rd Int. Workshop on Web Services and Formal Methods*, number 4184 in LNCS, pages 148–162. Springer, 2006.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (* WSDL*) 1.1*, 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[CGP09] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A Theory of Contracts for Web Services. *ACM Transactions on Programming Languages and Systems*, 31(5), 2009.

[CHY07] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centered programming for web services. In *Proceedings of 16th European Symposium on Programming*, LNCS, 2007.

[CLP09] Samuele Carpineti, Cosimo Laneve, and Luca Padovani. PiDuce – A Project for Experimenting Web Services Technologies. *Science of Computer Programming*, 74(10):777–811, 2009.

[Cos95] R. Di Cosmo. *Isomorphisms of Types: from Lambda Calculus to Information Retrieval and Language Desig.* Birkhauser, 1995. ISBN-0-8176-3763-X.

[CP09] Giuseppe Castagna and Luca Padovani. Contracts for Mobile Processes. In *Proceedings of the 20th International Conference on Concurrency Theory (CONCUR'09)*, volume 5710 of *LNCS*, pages 211–228. Springer, 2009.

[CPS93] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The concurrency workbench: a semantics-based tool for the verification of concurrent systems. *ACM Trans. Program. Lang. Syst.*, 15(1):36–72, 1993.

[CRR02] Sagar Chaki, Sriram K. Rajamani, and Jakob Rehof. Types as models: model checking message-passing programs. *SIGPLAN Not.*, 37(1):45–57, 2002.

[DH84] Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theor. Comput. Sci*, 34:83–133, 1984.

[DH87] Rocco De Nicola and Matthew Hennessy. CCS without τ's. In *Proceedings of TAPSOFT'87/CAAP'87*, LNCS 249, pages 138–152. Springer, 1987.

[FL01] Cédric Fournet and Cosimo Laneve. Bisimulations in the join-calculus. *Theoretical Computer Science*, 266(1-2):569–603, 2001.

[GH05] Simon Gay and Malcolm Hole. Subtyping for session types in the π-calculus. *Acta Informatica*, 42(2-3):191–225, 2005.

[Hen88] M. Hennessy. *Algebraic Theory of Processes.* Foundation of Computing. MIT Press, 1988.

[Hon93] Kohei Honda. Types for dyadic interaction. In *CONCUR'93*, LNCS 715, pages 509–523. Springer, 1993.

[HP03] Haruo Hosoya and Benjamin C. Pierce. XDuce: A statically typed XML processing language. *ACM Trans. Internet Techn.*, 3(2):117–148, 2003.

[HVK98] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, LNCS 1381, pages 122–138. Springer, 1998.

[HYC08] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM, 2008.

[IK01] Atsushi Igarashi and Naoki Kobayashi. A generic type system for the pi-calculus. In *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 128–141. ACM, 2001.

[KBR+05] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. *Web Services Choreography Description Language 1.0*, 2005. http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/.

[LP07] Cosimo Laneve and Luca Padovani. The *must* preorder revisited – an algebraic theory for web services contracts. In *CONCUR'07*, LNCS 4703, pages 212–225. Springer, 2007.

[LP08] Cosimo Laneve and Luca Padovani. The pairing of contracts and session types. In *Concurrency, Graphs and Models*, volume 5065 of *Lecture Notes in Computer Science*, pages 681–700. Springer, 2008.

[LP13] Cosimo Laneve and Luca Padovani. An Algebraic Theory for Web Service Contracts. In *Proceedings of 10th International Conference on integrated Formal Methods*, volume LNCS 7940, pages 301–315. Springer, 2013.

[Mil82] Robin Milner. *A Calculus of Communicating Systems.* Springer, 1982.

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[NN94] Hanne Riis Nielson and Flemming Nielson. Higher-order concurrent programs with finite communication topology (extended abstract). In *Proceedings of POPL'94*, pages 84–97. ACM Press, 1994.

[OVvdA+07] Chun Ouyang, Eric Verbeek, Wil M.P. van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur H.M. ter

Hofstede. Formal semantics and analysis of control flow in ws-bpel. *Science of Computer Programming*, 67(2–3):162 – 198, 2007.

[Pad08] Luca Padovani. Contract-Directed Synthesis of Simple Orchestrators. In *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR'08)*, volume LNCS 5201, pages 131–146. Springer, 2008.

[Pad09] Luca Padovani. *Contract-Based Discovery and Adaptation of Web Services*, volume LNCS 5569, pages 213–260. Springer, 2009.

[Pad10] Luca Padovani. Contract-Based Discovery of Web Services Modulo Simple Orchestrators. *Theoretical Computer Science*, 411:3328–3347, 2010.

[Pad13] Luca Padovani. Fair Subtyping for Open Session Types. In *Proceedings of 40th International Colloquium on Automata, Languages, and Programming, Part II*, volume LNCS 7966, pages 373–384. Springer, 2013.

[Pad14] Luca Padovani. Fair Subtyping for Multi-Party Session Types. *Mathematical Structures in Computer Science*, pages 1–41, 2014.

[PS93] Andrew M. Pitts and Ian D. B. Stark. Observable properties of higher order functions that dynamically create local names, or what's new? In *18th International Symposium on Mathematical Foundations of Computer Science*, volume 711 of *Lecture Notes in Computer Science*, pages 122–141. Springer, 1993.

[Rit93] Mikael Rittri. Retrieving library functions by unifying types modulo linear isomorphism. *RAIRO Theoretical Informatics and Applications*, 27(6):523–540, 1993.

A. Proofs

Notation

In this section we use some additional yet fairly conventional notation.

- We let $\leq$ be the prefixing ordering relation between sequences of actions.
- We generalize the definition of $\mathsf{actions}(\cdot)$ to sequences of actions so that $\mathsf{actions}(\varphi)$ is the set of actions occurring in φ.
- We write $\overline{\varphi}$ for the sequence obtained from φ by swapping each action with the corresponding co-action.
- We write $\mathsf{X} \stackrel{\alpha_1\cdots\alpha_n}{\Longrightarrow}$ if there exists X' such that $\mathsf{X} \stackrel{\alpha_1}{\Longrightarrow} \cdots \stackrel{\alpha_n}{\Longrightarrow} \mathsf{X}'$.
- We extend continuations to sequences of actions. Let $\mathsf{X} \stackrel{\varphi}{\Longrightarrow}$. If $\varphi = \varepsilon$, then $\mathsf{X}(\varphi) = \mathsf{X}$; if $\varphi = \alpha\varphi'$, then $\mathsf{X}(\varphi) = \mathsf{X}(\alpha)(\varphi')$.
- We generalize the convergence and divergence predicates so that $\mathsf{X}{\downarrow}\varepsilon$ if $\mathsf{X}{\downarrow}$ and $\mathsf{X}{\downarrow}\alpha\varphi$ if $\mathsf{X}{\downarrow}$ and $\mathsf{X} \stackrel{\alpha}{\Longrightarrow} \mathsf{X}'$ implies $\mathsf{X}'{\downarrow}\varphi$. We write $\mathsf{X}{\uparrow}\varphi$ if not $\mathsf{X}{\downarrow}\varphi$.
- Many proofs rely on the "unzipping of derivations" [Hen88], which decomposes the interaction between two terms X and Y. In particular, let $\mathsf{X} \mid_{\mathsf{N}} \mathsf{Y} \stackrel{\varepsilon}{\Longrightarrow} \mathsf{X}' \mid_{\mathsf{N}} \mathsf{Y}'$. Then, by definition of $\mid_{\mathsf{N}}$, there is a sequence φ of actions such that $\mathsf{X} \stackrel{\varphi}{\Longrightarrow} \mathsf{X}'$ and $\mathsf{Y} \stackrel{\overline{\varphi}}{\Longrightarrow} \mathsf{Y}'$. By "zipping" we mean the inverse process whereby two derivations $\mathsf{X} \stackrel{\varphi}{\Longrightarrow} \mathsf{X}'$ and $\mathsf{Y} \stackrel{\overline{\varphi}}{\Longrightarrow} \mathsf{Y}'$ are combined to produce $\mathsf{X} \mid_{\mathsf{N}} \mathsf{Y} \stackrel{\varepsilon}{\Longrightarrow} \mathsf{X}' \mid_{\mathsf{N}} \mathsf{Y}'$. See [Hen88] for a more detailed discussion.

Proof of Lemma 2.1

The proof of Lemma 2.1 is a simple adaptation of a similar result for CCS$_*$ [BGZ09].

Lemma 2.1. *Let* $\mathsf{reach}(P) = \{Q \mid$ *there are* $\mu_1, \dots, \mu_n$ *with* $P \stackrel{\mu_1}{\longrightarrow} \cdots \stackrel{\mu_n}{\longrightarrow} Q\}$. *Then, for every activity* P, *the set* $\mathsf{reach}(P)$ *is always finite.*

Proof. For an arbitrary activity P we inductively define the set $\mathcal{D}(P)$ as follows:

$$
\begin{array}{rcl}
\mathcal{D}(\mathsf{0}) & \stackrel{\text{def}}{=} & \{\mathsf{0}\} \\
\mathcal{D}(\sum_{i\in I} \alpha_i \,;\, P_i) & \stackrel{\text{def}}{=} & \{\sum_{i\in I} \alpha_i \,;\, P_i\} \cup \bigcup_{i\in I} \mathcal{D}(P_i) \\
\mathcal{D}(P \mid_A Q) & \stackrel{\text{def}}{=} & \{P' \mid_A Q' \mid P' \in \mathcal{D}(P), Q' \in \mathcal{D}(Q)\} \\
\mathcal{D}(P \,;\, Q) & \stackrel{\text{def}}{=} & \{P' \,;\, Q \mid P' \in \mathcal{D}(P)\} \cup \mathcal{D}(Q) \\
\mathcal{D}(\bigoplus_{i\in I} P_i) & \stackrel{\text{def}}{=} & \{\bigoplus_{i\in I} P_i\} \cup \bigcup_{i\in I} \mathcal{D}(P_i) \\
\mathcal{D}(P^*) & \stackrel{\text{def}}{=} & \{P^*, \mathsf{0}\} \cup \{P' \,;\, P^* \mid P' \in \mathcal{D}(P)\}
\end{array}
$$

A simple inductive argument allows one to establish that $\mathcal{D}(P)$ is finite for every P. Now, we conclude if we are able to show that $P \xrightarrow{\mu} P'$ implies $\mathcal{D}(P') \subseteq \mathcal{D}(P)$. This follows from an induction on the derivation of $P \xrightarrow{\mu} P'$. We leave the details to the reader. $\square$

Proof of Theorem 4.1

Lemma A.1. *Let* $\mathsf{X}\ \mathcal{R}\ \mathsf{Y}$ *where* $\mathcal{R}$ *is a coinductive compliance and* $\mathsf{Y} \stackrel{\varphi}{\Longrightarrow}$*. Then either there exists* $\varphi' \leq \varphi$ *such that* $\mathsf{X}(\varphi')\uparrow$ *or* $\mathsf{X}(\varphi)\downarrow$ *and* $\mathsf{X}(\varphi)\ \mathcal{R}\ \mathsf{Y}(\varphi)$*.*

Proof. By induction on φ. If $\mathsf{X}\uparrow$, then we conclude immediately by taking $\varphi' = \varepsilon$. If $\mathsf{X}\downarrow$, then by definition of coinductive compliance we have $\mathsf{Y}\downarrow$. If $\varphi = \varepsilon$, then we conclude $\mathsf{X}(\varphi)\ \mathcal{R}\ \mathsf{Y}(\varphi)$. If $\varphi = \alpha\varphi''$, then by definition of coinductive compliance we have $\mathsf{X}' \stackrel{\text{def}}{=} \mathsf{X}(\alpha)\ \mathcal{R}\ \mathsf{Y}(\alpha) \stackrel{\text{def}}{=} \mathsf{Y}'$. By induction hypothesis we have that either there exists $\varphi''' \leq \varphi''$ such that $\mathsf{X}'(\varphi''')\uparrow$ or $\mathsf{X}'(\varphi'')\downarrow$ and $\mathsf{X}'(\varphi'')\ \mathcal{R}\ \mathsf{Y}'(\varphi'')$. In the first subcase we conclude by taking $\varphi' = \alpha\varphi'''$, because $\mathsf{X}(\varphi') = \mathsf{X}(\alpha\varphi''') = \mathsf{X}'(\varphi''')$. In the second subcase we conclude by observing that $\mathsf{X}(\varphi) = \mathsf{X}'(\varphi'')$ and $\mathsf{Y}(\varphi) = \mathsf{Y}'(\varphi'')$. $\square$

Theorem 4.1. *For every* X *and* Y*, the following statements are equivalent:*

1. $\mathsf{X} \preceq \mathsf{Y}$;
2. $\mathsf{X} \precsim \mathsf{Y}$;
3. $\mathsf{X} \precsim_C \mathsf{Y}$;
4. $\mathsf{X} \precsim_{A+C} \mathsf{Y}$.

Proof. We show $1 \Rightarrow 2$ and $2 \Rightarrow 1$, the remaining implications are analogous since the proof does not depend on the syntax of activities/behaviors except for the availability of prefixes, internal and external choices, which are valid constructs for both activities and behaviors.

- ($1 \Rightarrow 2$) Let $\mathsf{T} \dashv \mathsf{X}$ and consider a derivation of $\mathsf{Y} \mid_{\mathsf{N}} \mathsf{T} \stackrel{\varepsilon}{\Longrightarrow} \mathsf{Y}' \mid_{\mathsf{N}} \mathsf{T}'$. By unzipping this derivation we obtain a sequence φ of actions such that $\mathsf{T} \stackrel{\overline{\varphi}}{\Longrightarrow} \mathsf{T}'$ and $\mathsf{Y} \stackrel{\varphi}{\Longrightarrow} \mathsf{Y}'$. From Lemma A.1 we deduce that either there exists $\varphi' \leq \varphi$ such that $\mathsf{X}(\varphi')\uparrow$ or $\mathsf{X}(\varphi)\downarrow$ and $\mathsf{X}(\varphi) \preceq \mathsf{Y}(\varphi)$. In the first case, using the hypothesis $\mathsf{T} \dashv \mathsf{X}$ we conclude $\{\mathsf{e}\} = \mathsf{init}(\mathsf{T}(\overline{\varphi}')) = \mathsf{init}(\mathsf{T}')$. In the second case, suppose $\mathsf{Y}' \mid_{\mathsf{N}} \mathsf{T}' \stackrel{\varepsilon}{\not\longrightarrow}$. From the definition of coinductive compliance we have $\mathsf{Y}(\varphi)\downarrow$ and, from condition (2) of Definition 4.1, we know that there exists X' such that $\mathsf{X}(\varphi) \stackrel{\varepsilon}{\Longrightarrow} \mathsf{X}' \stackrel{\varepsilon}{\not\longrightarrow}$ and $\mathsf{init}(\mathsf{X}') \subseteq \mathsf{init}(\mathsf{Y}')$. Then $\mathsf{X} \mid_{\mathsf{N}} \mathsf{T} \stackrel{\varepsilon}{\Longrightarrow} \mathsf{X}' \mid_{\mathsf{N}} \mathsf{T}' \stackrel{\varepsilon}{\not\longrightarrow}$ and, using the hypothesis $\mathsf{T} \dashv \mathsf{X}$, we conclude $\{\mathsf{e}\} \subseteq \mathsf{init}(\mathsf{T}')$.
- ($2 \Rightarrow 1$) Suppose $\mathsf{X} \precsim \mathsf{Y}$ and $\mathsf{X}\downarrow$. Regarding condition (1) of Definition 4.1, suppose by contradiction $\mathsf{Y}\uparrow$, let a be a name that does not occur in X nor in Y and consider $\mathsf{T} \stackrel{\text{def}}{=} \mathsf{e} + a$. Then $\mathsf{T} \dashv \mathsf{X}$ but $\mathsf{T} \not\dashv \mathsf{Y}$, which contradicts the hypothesis $\mathsf{X} \precsim \mathsf{Y}$. Hence $\mathsf{Y}\downarrow$ and condition (1) is satisfied. Regarding condition (2) of Definition 4.1, let $\mathrm{S}_1, \dots, \mathrm{S}_n$ be the ready sets of X (there are finitely many of them) and suppose that there exists R such that $\mathsf{Y} \Downarrow \mathrm{R}$ and $\mathrm{S}_i \not\subseteq \mathrm{R}$ for every $1 \leq i \leq n$, that is there exists $\alpha_i \in \mathrm{R}_i \setminus \mathrm{S}$ for every $1 \leq i \leq n$. Consider $\mathsf{T} \stackrel{\text{def}}{=} \sum_{i=1}^{n} \overline{\alpha}_i\,;\mathsf{e}$. We have $\mathsf{T} \dashv \mathsf{X}$ and $\mathsf{T} \not\dashv \mathsf{Y}$, which contradicts the hypothesis $\mathsf{X} \precsim \mathsf{Y}$. Hence there exists $1 \leq i \leq n$ such that $\mathrm{S}_i \subseteq \mathrm{R}$ and condition (2) is satisfied. Regarding condition (3) of Definition 4.1, suppose $\mathsf{Y} \stackrel{\alpha}{\Longrightarrow}$ and suppose, by contradiction, that $\mathsf{X} \stackrel{\alpha}{\not\Longrightarrow}$. Then $\mathsf{e} + \overline{\alpha} \dashv \mathsf{X}$ and $\mathsf{e} + \overline{\alpha} \not\dashv \mathsf{Y}$, which contradicts the hypothesis $\mathsf{X} \precsim \mathsf{Y}$, hence $\mathsf{X} \stackrel{\alpha}{\Longrightarrow}$. Now let T' be an arbitrary activity/behavior such that $\mathsf{T}' \dashv \mathsf{X}(\alpha)$ and consider $\mathsf{T} \stackrel{\text{def}}{=} \mathsf{e} + \overline{\alpha}\,;\mathsf{T}'$. We have $\mathsf{T} \dashv \mathsf{X}$ hence, from the hypothesis $\mathsf{X} \precsim \mathsf{Y}$, we deduce $\mathsf{T} \dashv \mathsf{Y}$. This implies $\mathsf{T}' \dashv \mathsf{Y}(\alpha)$, hence we conclude $\mathsf{X}(\alpha) \precsim \mathsf{Y}(\alpha)$ because T' is arbitrary, and condition (3) is satisfied. $\square$

Proof of Theorem 3.1

Theorem 3.1. $P \approx \mathsf{C}_P$.

Proof. By Theorem 4.1 it is sufficient to prove that $\mathcal{R} \stackrel{\text{def}}{=} \{(P, \mathsf{C}_P)\}$ is a coinductive compliance. The proof that $\mathcal{R}^{-1}$ is also a coinductive compliance is similar. Let $\mathsf{X}\ \mathcal{R}\ \mathsf{Y}$. Then $\mathsf{X} = P$ and $\mathsf{Y} = \mathsf{C}_P$ for some P. Suppose $P \Downarrow$ for otherwise there is nothing to prove. From the definition of C_P we deduce $\mathsf{C}_P \Downarrow$, hence condition (1) of Definition 4.1 is satisfied. Now let $\mathsf{C}_P \Downarrow \mathrm{R}$. By definition of C_P we have $P \Downarrow \mathrm{S}$ with $\mathrm{S} \subseteq \mathrm{R}$, therefore condition (2) of Definition 4.1 is satisfied. Finally, suppose $\mathsf{C}_P \stackrel{\alpha}{\Longrightarrow}$. Then $P \stackrel{\alpha}{\Longrightarrow}$. By definition of $\mathcal{R}$ we have $P(\alpha)\ \mathcal{R}\ \mathsf{C}_{P(\alpha)}$ and we conclude that condition (3) of Definition 4.1 is satisfied by observing that $\mathsf{C}_P(\alpha) = \mathsf{C}_{P(\alpha)}$. □

Proof of Theorem 4.2

Theorem 4.2. $\mathsf{X} \sqsubseteq_{\mathtt{must}} \mathsf{Y}$ *if and only if* $\mathsf{X} \lesssim \mathsf{Y}$.

Proof. Because of Theorem 4.1 we can show the equivalence between $\sqsubseteq_{\mathtt{must}}$ and $\preceq$.
($\Leftarrow$) Let $\mathsf{X} \preceq \mathsf{Y}$ and assume, by contradiction, that $\mathsf{X}\ \mathtt{must}\ \mathsf{T}$ and $\mathsf{Y}\ \not{\mathtt{must}}\ \mathsf{T}$ for some T. Then there must be a maximal computation $\mathsf{Y} \mid_{\mathsf{N}} \mathsf{T} = \mathsf{Y}_0 \mid_{\mathsf{N}} \mathsf{T}_0 \stackrel{\varepsilon}{\longrightarrow} \mathsf{Y}_1 \mid_{\mathsf{N}} \mathsf{T}_1 \stackrel{\varepsilon}{\longrightarrow} \cdots$ such that $\mathsf{T}_i \stackrel{\mathsf{e}}{\not\longrightarrow}$ for every $i = 0, 1, \ldots$. We distinguish two cases: (a) the computation is finite, (b) the computation is infinite.

In case (a) there exists n such that $\mathsf{Y}_n \mid_{\mathsf{N}} \mathsf{T}_n \stackrel{\varepsilon}{\not\longrightarrow}$. Then there exists φ such that $\mathsf{Y} \stackrel{\varphi}{\Longrightarrow} \mathsf{Y}_n$ and $\mathsf{T} \stackrel{\overline{\varphi}}{\Longrightarrow} \mathsf{T}_n$. From Lemma A.1 we deduce that either there exist $\varphi' \leq \varphi$ and X' such that $\mathsf{X} \stackrel{\varphi'}{\Longrightarrow} \mathsf{X}'\!\uparrow$ or $\mathsf{X}(\varphi)\!\downarrow$ and $\mathsf{X}(\varphi) \preceq \mathsf{Y}(\varphi)$. By zipping the computations starting from X and T, in the first subcase we can build an infinite computation $\mathsf{X} \mid_{\mathsf{N}} \mathsf{T} \stackrel{\varepsilon}{\Longrightarrow} \mathsf{X}' \mid_{\mathsf{N}} \mathsf{T}_{|\varphi'|} \stackrel{\varepsilon}{\longrightarrow} \cdots$, while in the second case we can find an X' such that $\mathsf{X} \stackrel{\varphi}{\Longrightarrow} \mathsf{X}' \stackrel{\varepsilon}{\not\longrightarrow}$ and $\mathsf{init}(\mathsf{X}') \subseteq \mathsf{init}(\mathsf{Y}_n)$. In both cases we deduce $\mathsf{X}\ \not{\mathtt{must}}\ \mathsf{T}$, which is absurd.

In case (b), we distinguish two subcases:

b1. there exists n such that $\mathsf{Y}_n\!\uparrow$ or $\mathsf{T}_n\!\uparrow$. Then using an argument similar to case (a), it is possible to show a contradiction for $\mathsf{X}\ \not{\mathtt{must}}\ \mathsf{T}$.

b2. Y and T communicate infinitely often, that is the computation may be unzipped into $\mathsf{Y} \stackrel{\varphi}{\Longrightarrow}$ and $\mathsf{T} \stackrel{\overline{\varphi}}{\Longrightarrow}$, where φ is infinite. It is easy to prove that, for every finite $\varphi' \leq \varphi$, there is X' such that $\mathsf{X} \stackrel{\varphi'}{\Longrightarrow} \mathsf{X}'$. Therefore there exists an infinite computation of $\mathsf{X} \mid_{\mathsf{N}} \mathsf{T}$ that transits in the same states $\mathsf{T}_0, \mathsf{T}_1, \ldots$ as the ones of $\mathsf{Y} \mid_{\mathsf{N}} \mathsf{T}$. This contradicts the hypothesis $\mathsf{X}\ \mathtt{must}\ \mathsf{T}$.

($\Rightarrow$) We prove that

$$\mathcal{R} \stackrel{\text{def}}{=} \{(\mathsf{Y}_1, \mathsf{Y}_2) \mid \mathsf{Y}_1 \sqsubseteq_{\mathtt{must}} \mathsf{Y}_2\}$$

is a coinductive compliance. Let $\mathsf{Y}_1\ \mathcal{R}\ \mathsf{Y}_2$ and $\mathsf{Y}_1\!\downarrow$. We prove the three conditions of Definition 4.1 in order.

1. Suppose by contradiction $\mathsf{Y}_2\!\uparrow$. Then $\mathsf{Y}_1\ \mathtt{must}\ \mathsf{e} \oplus \mathsf{e}$ whereas $\mathsf{Y}_2\ \not{\mathtt{must}}\ \mathsf{e} \oplus \mathsf{e}$ which is absurd, hence $\mathsf{Y}_2\!\downarrow$.
2. Let $\mathrm{R}_1, \ldots, \mathrm{R}_n$ be the ready sets of Y_1. Assume by contradiction that there exists S such that $\mathsf{Y}_2 \Downarrow \mathrm{S}$ and $\mathrm{R}_i \not\subseteq \mathrm{R}$ for every $1 \leq i \leq n$. That is, every R_i is nonempty and, for every $1 \leq i \leq n$, there exists $\alpha_i \in \mathrm{R}_i \setminus \mathrm{S}$. Now $\mathsf{Y}_1\ \mathtt{must}\ \sum_{1 \leq i \leq n} \overline{\alpha_i}\,;\mathsf{e}$ while $\mathsf{Y}_2\ \not{\mathtt{must}}\ \sum_{1 \leq i \leq n} \overline{\alpha_i}\,;\mathsf{e}$, which is absurd.
3. Let $\mathsf{Y}_2 \stackrel{\alpha}{\Longrightarrow} \mathsf{Y}_2'$. Then also $\mathsf{Y}_1 \stackrel{\alpha}{\Longrightarrow}$. In fact, if this were not the case, then $\mathsf{Y}_1\ \mathtt{must}\ \mathsf{e} + \overline{\alpha}$ while $\mathsf{Y}_2\ \not{\mathtt{must}}\ \mathsf{e} + \overline{\alpha}$, which is absurd. Let T be an arbitrary term such that $\mathsf{Y}_1(\alpha)\ \mathtt{must}\ \mathsf{T}$. Then $\mathsf{Y}_1\ \mathtt{must}\ \mathsf{e} + \overline{\alpha}\,;\mathsf{T}$. From the hypothesis $\mathsf{Y}_1 \sqsubseteq_{\mathtt{must}} \mathsf{Y}_2$ we deduce $\mathsf{Y}_2\ \mathtt{must}\ \mathsf{e} + \overline{\alpha}\,;\mathsf{T}$, therefore $\mathsf{Y}_2(\alpha)\ \mathtt{must}\ \mathsf{T}$. We conclude $\mathsf{Y}_1(\alpha)\ \mathcal{R}\ \mathsf{Y}_2(\alpha)$ by definition of $\mathcal{R}$. □

Proof of Theorem 5.1

Lemma A.2. *Let* $\mathrm{I} : \sigma\ \mathcal{R}\ \mathrm{J} : \tau$ *where* $\mathcal{R}$ *is a coinductive compliance and* $\tau \stackrel{\varphi}{\Longrightarrow}$ *and* $\mathsf{actions}(\varphi) \subseteq \mathrm{I}$. *Then either there exists* $\varphi' \leq \varphi$ *such that* $\sigma(\varphi')\!\uparrow$ *or* $\sigma(\varphi)\!\downarrow$ *and* $\sigma(\varphi)\ \mathcal{R}\ \tau(\varphi)$.

Proof. Analogous to that of Lemma A.1. □

Theorem 5.1. $\lesssim$ *is the largest coinductive subcontract relation.*

Proof. We begin showing that $\lesssim$ is a coinductive subcontract relation. Suppose $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau$ and $\sigma\!\downarrow$. Then by Definition 5.1 we know $\mathrm{I} \subseteq \mathrm{J}$. We now prove the conditions of Definition 5.2 in order.

1. Suppose by contradiction that $\tau\uparrow$. Then $\Omega \dashv \sigma$ and $\Omega \not\dashv \tau$, which contradicts the hypothesis $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau$, hence we conclude $\tau\downarrow$ and condition (1) is satisfied.
2. Let $\mathrm{R}_1, \dots, \mathrm{R}_n$ be the ready sets of σ and assume by contradiction that there exists S such that $\tau \Downarrow \mathrm{S}$ and for every $1 \leq i \leq n$ there exists $\alpha_i \in \mathrm{R}_i \setminus \mathrm{S}$. By definition of ready set we have $\tau \stackrel{\varepsilon}{\Longrightarrow} \tau' \stackrel{\varepsilon}{\not\longrightarrow}$ and $\mathsf{init}(\tau') \subseteq \mathrm{S}$. Consider $\rho \stackrel{\text{def}}{=} \sum_{1 \leq i \leq n} \overline{\alpha_i} \,;\mathsf{e}$. Then, $\rho \dashv \sigma$ but $\rho \not\dashv \tau$ because $\tau \mid_{\mathsf{N}} \rho \stackrel{\varepsilon}{\Longrightarrow} \tau' \mid_{\mathsf{N}} \rho \stackrel{\varepsilon}{\not\longrightarrow}$ and $\mathsf{e} \notin \mathsf{init}(\rho)$, which is absurd. Hence we conclude that $\mathrm{R}_i \subseteq \mathrm{S}$ for some $1 \leq i \leq n$ and condition (2) is satisfied.
3. Let $\tau \stackrel{\alpha}{\Longrightarrow}$ and $\alpha \in \mathrm{I}$. It must be the case that $\sigma \stackrel{\alpha}{\Longrightarrow}$, otherwise $\mathsf{e}+\overline{\alpha} \dashv \sigma$ while $\mathsf{e}+\overline{\alpha} \not\dashv \tau$, which contradicts the hypothesis $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau$. Let ρ be an arbitrary behavior such that $\mathsf{actions}(\rho) \setminus \{\mathsf{e}\} \subseteq \overline{\mathrm{I}}$ and $\rho \dashv \sigma(\alpha)$. Then $\mathsf{e}+\overline{\alpha}\,;\rho \dashv \sigma$. From the hypothesis $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau$ we deduce $\mathsf{e}+\overline{\alpha}\,;\rho \dashv \tau$, hence $\rho \dashv \tau(\alpha)$. We conclude $\mathrm{I} : \sigma(\alpha) \lesssim \mathrm{J} : \tau(\alpha)$ because ρ is arbitrary, hence condition (3) is satisfied.

Next we show that every coinductive subcontract relation is included in $\lesssim$, proving that $\lesssim$ is indeed the largest one. Let $\mathrm{I} : \sigma \,\mathcal{R}\, \mathrm{J} : \tau$ where $\mathcal{R}$ is a coinductive subcontract. By Definition 5.2 we know that $\mathrm{I} \subseteq \mathrm{J}$. Let $\mathrm{K} : \rho$ be such that $\mathrm{K} \setminus \{\mathsf{e}\} \subseteq \overline{\mathrm{I}}$ and $\rho \dashv \sigma$. Consider a derivation of $\tau \mid_{\mathsf{N}} \rho \stackrel{\varepsilon}{\Longrightarrow} \tau' \mid_{\mathsf{N}} \rho'$. By unzipping this derivation we obtain a sequence φ of actions such that $\rho \stackrel{\overline{\varphi}}{\Longrightarrow} \rho'$ and $\tau \stackrel{\varphi}{\Longrightarrow} \tau'$ and furthermore $\mathsf{actions}(\varphi) \subseteq \mathrm{I}$. From Lemma A.2 we deduce that either there exists $\varphi' \leq \varphi$ such that $\sigma(\varphi')\uparrow$ or $\sigma(\varphi)\downarrow$ and $\sigma(\varphi) \,\mathcal{R}\, \tau(\varphi)$. In the first case, using the hypothesis $\rho \dashv \sigma$ we conclude $\{\mathsf{e}\} = \mathsf{init}(\rho(\overline{\varphi}')) = \mathsf{init}(\rho')$. In the second case, suppose $\tau' \mid_{\mathsf{N}} \rho' \stackrel{\varepsilon}{\not\longrightarrow}$. From the definition of coinductive subcontract we have $\tau(\varphi)\downarrow$ and, from condition (2) of Definition 5.2, we know that there exists σ' such that $\sigma(\varphi) \stackrel{\varepsilon}{\Longrightarrow} \sigma' \stackrel{\varepsilon}{\not\longrightarrow}$ and $\mathsf{init}(\sigma') \subseteq \mathsf{init}(\tau')$. Then $\sigma \mid_{\mathsf{N}} \rho \stackrel{\varepsilon}{\Longrightarrow} \sigma' \mid_{\mathsf{N}} \rho' \stackrel{\varepsilon}{\not\longrightarrow}$ and, using the hypothesis $\rho \dashv \sigma$, we conclude $\{\mathsf{e}\} \subseteq \mathsf{init}(\rho')$. ☐

Proofs of Propositions 5.1 and 5.2

Proposition 5.1. *The following properties hold:*

1. *If* $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau$ *and* $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau'$, *then* $\mathrm{I} : \sigma \lesssim \mathrm{J} : \tau \oplus \tau'$;
2. *if* $\mathrm{I} : \mathsf{0} \lesssim \mathrm{J} : \tau$, *then* $\mathrm{I} : \sigma \lesssim \mathrm{J} : \sigma + \tau$ *(*width extension*)*;
3. *if* $\mathrm{I} : \mathsf{0} \lesssim \mathrm{J} : \tau$, *then* $\mathrm{I} : \sigma \lesssim \mathrm{J} : \sigma\{^{\tau}/_{\mathsf{0}}\}$, *where* $\sigma\{^{\tau}/_{\mathsf{0}}\}$ *is the replacement of every occurrence of the contract name* $\mathsf{0}$ *with* τ *(*depth extension*)*.

Proof. We only show the proof of item (2), the others being simpler/analogous. Using Theorem 5.1, it is enough to show that

$$\mathcal{R} \stackrel{\text{def}}{=} \{(\mathrm{I} : \sigma, \mathrm{J} : \sigma + \tau) \mid \mathrm{I} : \mathsf{0} \lesssim \mathrm{J} : \tau\} \cup \{(\mathrm{I} : \sigma, \mathrm{I} : \sigma) \mid \mathrm{I} : \sigma \text{ is an extended contract}\}$$

is a coinductive subcontract. Since $\lesssim$ is obviously reflexive, the only interesting case to consider is when $\mathrm{I} : \sigma \,\mathcal{R}\, \mathrm{J} : \sigma+\tau$ and $\mathrm{I} : \mathsf{0} \lesssim \mathrm{J} : \tau$. From $\mathrm{I} : \mathsf{0} \lesssim \mathrm{J} : \tau$ we deduce $\mathrm{I} \subseteq \mathrm{J}$. Now suppose $\sigma\downarrow$; we prove the conditions of Definition 5.2 in order:

1. From $\mathrm{I} : \mathsf{0} \lesssim \mathrm{J} : \tau$ we deduce $\tau\downarrow$, hence $\sigma + \tau\downarrow$.
2. Let $\sigma + \tau \Downarrow \mathrm{R}$. Then there exist R_1 and R_2 such that $\sigma \Downarrow \mathrm{R}_1$ and $\tau \Downarrow \mathrm{R}_2$ and $\mathrm{R} = \mathrm{R}_1 \cup \mathrm{R}_2$. We conclude by observing that $\mathrm{R}_1 \subseteq \mathrm{R}$.
3. Let $\sigma + \tau \stackrel{\alpha}{\Longrightarrow}$ and $\alpha \in \mathrm{I}$. From $\mathrm{I} : \mathsf{0} \lesssim \mathrm{J} : \tau$ we deduce $\tau \stackrel{\alpha}{\not\Longrightarrow}$, hence $(\sigma + \tau)(\alpha) = \sigma(\alpha)$. We conclude $\sigma(\alpha) \,\mathcal{R}\, \sigma(\alpha)$ by definition of $\mathcal{R}$. ☐

Proposition 5.2. $\mathrm{I} : \sigma \approx \mathrm{J} : \tau$ *if and only if* $\sigma \eqsim \tau$ *and* $\mathrm{I} = \mathrm{J}$.

Proof. Using Theorems 4.1 and 5.1 it is enough to show that

$$\mathcal{R}_1 \stackrel{\text{def}}{=} \{(\sigma, \tau) \mid \mathrm{I} : \sigma \lesssim \mathrm{I} : \tau\} \qquad \text{and} \qquad \mathcal{R}_2 \stackrel{\text{def}}{=} \{(\mathrm{I} : \sigma, \mathrm{I} : \tau) \mid \sigma \preceq \tau\}$$

respectively are a coinductive compliance and a coinductive subcontract. The result follows easily from Definitions 4.1 and 5.2. ☐

Proof of Theorem 6.1

Theorem 6.1. *Let* $\mathrm{K} : \rho$ *be a canonical extended contract. Then:*

1. $\rho \dashv \mathsf{D}^{\mathrm{K}}_{\rho}$;
2. *if* $\overline{\mathrm{K}} \setminus \{\bar{\mathsf{e}}\} \subseteq \mathrm{S}$ *and* $\rho \dashv \sigma$, *then* $\overline{\mathrm{K}} \setminus \{\bar{\mathsf{e}}\} : \mathsf{D}^{\mathrm{K}}_{\rho} \lesssim \mathrm{S} : \sigma$.

Proof. Regarding item 1, we remark that, by definition of dual, every derivation $\mathsf{D}^{\mathrm{K}}_{\rho} \mid_{\mathsf{N}} \rho \stackrel{\varepsilon}{\Longrightarrow} \sigma \mid_{\mathsf{N}} \rho'$ may be rewritten into $\mathsf{D}^{\mathrm{K}}_{\rho} \mid_{\mathsf{N}} \rho \stackrel{\varepsilon}{\Longrightarrow} \mathsf{D}^{\mathrm{K}}_{\rho(\varphi)} \mid_{\mathsf{N}} \rho \stackrel{\varepsilon}{\Longrightarrow} \sigma \mid_{\mathsf{N}} \rho'$, where $\rho(\varphi) \stackrel{\varepsilon}{\Longrightarrow} \rho'$ and $\mathsf{D}^{\mathrm{K}}_{\rho(\varphi)} \stackrel{\varepsilon}{\Longrightarrow} \sigma$.

If $\sigma\!\uparrow$, then $\mathsf{D}^{\mathrm{K}}_{\rho(\varphi)} = \Omega$, which means that $\{\mathsf{e}\} = \mathsf{init}(\rho')$. In this case, the conditions in Definition 2.2 are satisfied. If $\sigma \mid_{\mathsf{N}} \rho' \stackrel{\varepsilon}{\not\longrightarrow}$, then assume by contradiction that $\mathsf{e} \notin \mathsf{init}(\rho')$. By definition of canonical client, $\mathsf{init}(\rho') \neq \varnothing$. Therefore, by definition of dual, $\mathsf{D}^{\mathrm{K}}_{\rho(\varphi)} \Downarrow \mathrm{R}$ implies $\mathrm{R} \neq \varnothing$ because $\mathsf{D}^{\mathrm{K}}_{\rho(\varphi)}$ has an empty ready set provided every ready set of $\rho(\varphi)$ contains e, which is not the case by hypothesis. Hence we conclude $\mathsf{init}(\sigma) \neq \varnothing$ and $\overline{\mathsf{init}(\rho')} \cap \mathsf{init}(\sigma) \neq \varnothing$ by definition of dual, which is absurd by $\sigma \mid_{\mathsf{N}} \rho' \stackrel{\varepsilon}{\not\longrightarrow}$.

Regarding item 2, let $\mathrm{K}' \stackrel{\text{def}}{=} \overline{\mathrm{K}} \setminus \{\bar{\mathsf{e}}\}$ and let $\mathcal{R}$ be the least relation such that:

- if $\sigma \stackrel{\varphi}{\Longrightarrow}$, $\rho \stackrel{\overline{\varphi}}{\Longrightarrow}$, and $\sigma{\downarrow}\varphi$, then $\mathrm{K}' : \mathsf{D}^{\mathrm{K}}_{\rho(\overline{\varphi})} \mathrel{\mathcal{R}} \mathrm{S} : \sigma(\varphi)$;
- if $\sigma \stackrel{\varphi}{\Longrightarrow}$ and either $\rho \stackrel{\overline{\varphi}}{\not\Longrightarrow}$ or $\sigma{\uparrow}\varphi$, then $\mathrm{K}' : \Omega \mathrel{\mathcal{R}} \mathrm{S} : \sigma(\varphi)$.

Note that $\mathrm{K}' : \mathsf{D}^{\mathrm{K}}_{\rho} \mathrel{\mathcal{R}} \mathrm{S} : \sigma$. Indeed, if $\sigma\!\uparrow$, then from $\rho \dashv \sigma$ we derive $\mathsf{init}(\rho) = \{\mathsf{e}\}$, hence $\mathsf{D}^{\mathrm{K}}_{\rho} = \Omega$ by definition of dual. Using Theorem 5.1 it suffices to prove that $\mathcal{R}$ is a coinductive subcontract. Let $\mathrm{K}' : \mathsf{D}^{\mathrm{K}}_{\rho(\overline{\varphi})} \mathrel{\mathcal{R}} \mathrm{S} : \sigma(\varphi)$ and $\mathsf{D}^{\mathrm{K}}_{\rho(\overline{\varphi})}{\downarrow}$. The conditions of Definition 5.2 are proved in order:

1. By definition of $\mathcal{R}$ we have $\sigma{\downarrow}\varphi$, hence $\sigma(\varphi){\downarrow}$.
2. Assume $\sigma(\varphi) \Downarrow \mathrm{R}$. Let $\{\mathrm{S}_1, \dots, \mathrm{S}_n\} \stackrel{\text{def}}{=} \{\mathrm{S} \mid \rho(\overline{\varphi}) \Downarrow \mathrm{S}, \mathsf{e} \notin \mathrm{S}\}$. From $\rho \dashv \sigma$ we derive $\mathrm{S}_i \cap \overline{\mathrm{R}} \neq \varnothing$ for every $1 \le i \le n$, namely there exists $\alpha_i \in \mathrm{S}_i \cap \overline{\mathrm{R}}$ for every $1 \le i \le n$. By definition of dual we have $\mathsf{D}^{\mathrm{K}}_{\rho(\overline{\varphi})} \Downarrow \{\overline{\alpha_1}, \dots, \overline{\alpha_n}\}$ and we conclude by observing that $\{\overline{\alpha_1}, \dots, \overline{\alpha_n}\} \subseteq \mathrm{R}$.
3. Let $\sigma(\varphi) \stackrel{\alpha}{\Longrightarrow}$ and $\alpha \in \mathrm{K}'$. Then $\overline{\alpha} \in \mathrm{K}$ and $\mathsf{D}^{\mathrm{K}}_{\rho(\overline{\varphi})} \stackrel{\alpha}{\Longrightarrow}$ by definition of dual. If $\rho(\overline{\varphi}) \stackrel{\overline{\alpha}}{\not\Longrightarrow}$, then $\mathsf{D}^{\mathrm{K}}_{\rho(\overline{\varphi})}(\alpha) = \Omega$ and we conclude $\mathrm{K}' : \Omega \mathrel{\mathcal{R}} \sigma(\varphi\alpha)$ by definition of $\mathcal{R}$. If $\rho(\overline{\varphi}) \stackrel{\overline{\alpha}}{\Longrightarrow}$, then we distinguish two subcases: either (i) $\sigma(\varphi\alpha){\uparrow}$ or (ii) $\sigma(\varphi\alpha){\downarrow}$. In subcase (i), from $\rho \dashv \sigma$ we derive $\mathsf{init}(\rho(\overline{\varphi\alpha})) = \{\mathsf{e}\}$, hence $\mathsf{D}^{\mathrm{K}}_{\rho(\overline{\varphi\alpha})} = \Omega$ and $\mathrm{K}' : \Omega \mathrel{\mathcal{R}} \mathrm{S} : \sigma(\varphi\alpha)$ by definition of $\mathcal{R}$. In subcase (ii) we have $\mathsf{D}^{\mathrm{K}}_{\rho(\overline{\varphi})}(\alpha) = \mathsf{D}^{\mathrm{K}}_{\rho(\overline{\varphi\alpha})}$ and we conclude $\mathrm{K}' : \mathsf{D}^{\mathrm{K}}_{\rho(\varphi)}(\alpha) \mathrel{\mathcal{R}} \mathrm{S} : \sigma(\varphi\alpha)$ by definition of $\mathcal{R}$. □

Proof of Theorem 7.1

Theorem 7.1. *Let* $\mathrm{K} : \rho$ *be compliant with* Γ_1 *and* Γ_2 *be a refinement of* Γ_1. *Then* $\mathrm{K} : \rho$ *is also compliant with* Γ_2.

Proof. Let $\Gamma_1 = \prod^{A}(\mathrm{I}_1 : \sigma_1, \dots, \mathrm{I}_n : \sigma_n)$ and $\Gamma_2 = \prod^{A}(\mathrm{J}_1 : \tau_1, \dots, \mathrm{J}_n : \tau_n)$ and consider a computation $\Gamma_2 \mid_{\mathsf{N}} \rho \stackrel{\varepsilon}{\Longrightarrow} \Gamma_2' \mid_{\mathsf{N}} \rho'$ where $\Gamma_2' = \prod^{A}(\mathrm{J}_1 : \tau_1', \dots, \mathrm{J}_n : \tau_n')$. By unzipping this computation we deduce that there exists a sequence φ of actions such that $\rho \stackrel{\overline{\varphi}}{\Longrightarrow} \rho'$ and $\Gamma_2 \stackrel{\varphi}{\Longrightarrow} \Gamma_2'$. By unzipping the computation of Γ_2 with respect to all of its participants we obtain n sequences $\varphi_1, \dots, \varphi_n$ of actions such that $\tau_i \stackrel{\varphi_i}{\Longrightarrow} \tau_i'$ for every $1 \le i \le n$. Note that φ is obtained by erasing pairs of complementary actions from the φ_i's, which correspond to synchronizations occurred *within* the choreography and solely pertain to names in A, and by suitably interleaving the remaining actions. Using $\mathrm{I}_i : \sigma_i \lesssim \mathrm{J}_i : \tau_i$, condition (2), and $\mathsf{actions}(\varphi) \subseteq \overline{\mathrm{K}} \setminus \{\bar{\mathsf{e}}\} \subseteq \mathsf{actions}(\Gamma_1)$, we deduce that all the actions in the φ_i are in I_i and $\sigma_i \stackrel{\varphi_i}{\Longrightarrow}$ for every $1 \le i \le n$. By zipping these derivations we obtain that $\Gamma_1 \stackrel{\varphi}{\Longrightarrow}$ as well.

We proceed by considering the two possibilities in Definition 2.2.

Suppose $\Gamma_2' \mid_{\mathsf{N}} \rho' \stackrel{\varepsilon}{\not\longrightarrow}$. If there exists Γ_1' such that $\Gamma_1 \stackrel{\varphi}{\Longrightarrow} \Gamma_1'$ and $\Gamma_1'\!\uparrow$, then from $\rho \dashv \Gamma_1$ we conclude

$\{\mathsf{e}\} = \mathsf{init}(\rho')$. If $\Gamma'_1\downarrow$ whenever $\Gamma_1 \stackrel{\varphi}{\Longrightarrow} \Gamma'_1$, then from $\Gamma_2 \stackrel{\varepsilon}{\not\longrightarrow}$ we deduce that for every $1 \leq i \leq n$ there exists σ'_i such that $\sigma_i \stackrel{\varphi_i}{\Longrightarrow} \sigma'_i$ and $\mathsf{init}(\sigma'_i) \subseteq \mathsf{init}(\tau'_i)$. Take $\Gamma'_1 = \prod^A(\mathrm{I}_1 : \sigma'_1, \dots, \mathrm{I}_n : \sigma'_n)$. Then $\Sigma_1 \mid_{\mathsf{N}} \stackrel{\varepsilon}{\Longrightarrow} \Sigma'_1 \mid_{\mathsf{N}} \rho' \stackrel{\varepsilon}{\not\longrightarrow}$ and from $\rho \dashv \Sigma_1$ we conclude $\{\mathsf{e}\} \subseteq \mathsf{init}(\rho')$.

Suppose $\Gamma'_2\uparrow$. This may happen either because one (or more) participants diverge autonomously, or because two (or more) participants interact infinitely often. By definition of refinement and using the same arguments as above, we obtain that there exists Γ'_1 such that $\Gamma_1 \stackrel{\varphi}{\Longrightarrow} \Gamma'_1$ and $\Gamma'_1\uparrow$. From the hypothesis $\rho \dashv \Gamma_1$ we conclude $\{\mathsf{e}\} = \mathsf{init}(\rho')$. □