A Theory of Contracts for Web Services
(work in progress)

Giuseppe Castagna
PPS (CNRS)
Université Paris 7
Paris, France

Nils Gesbert
LRI (CNRS)
Université Paris-Sud
Orsay, France

Luca Padovani
ISTI
Università degli Studi di Urbino
Urbino, Italy

ABSTRACT

We report preliminary results on our attempt to devise a type theory to decribe the detailed behaviour of web services and relate them. Our goal is to devise a type system that is as much minimal and language neutral as possible. We outline the possible practical impact of such a work, and the perspectives of future research it opens.

1. INTRODUCTION

The recent trend in Web Services is fostering a computing scenario where clients must be able to search at run-time services that provide some given capabilities. This scenario requires web-services to publish their capabilities in some known repository and the availability of powerful search operations for capabilities. Possible capabilities that one would like to search concern the format of the exchanged messages, and the protocol—or *contract*—required to interact successfully with the service.

The Web Service Description Language (WSDL) [10, 9, 8] provides a standardised technology for describing the interface exposed by a service. Such a description includes the service location, the format (or *schema*) of the exchanged messages, the transfer mechanism to be used (i.e. SOAP-RPC, or others), and the contract. In WSDL, contracts are basically limited to one-way (asynchronous) and request/response (synchronous) interactions. The Web Service Conversation Language (WSCL) [1] extends WSDL contracts by allowing the description of arbitrary, possibly cyclic sequences of exchanged messages between communicating parties.

Both WSDL and WSCL documents can be published in repositories [2, 11] so that they can be searched and queried. However, this immediately poses an issue related to the *compatibility* between different published contracts. It is necessary to define precise notions of contract similarity and compatibility and use them to perform service discovery in the same way as, say, type isomorphisms are used to perform library searches [19, 12]. Unfortunately, neither WSDL nor WSCL can effectively define these notions, for the very simple reason that they do not provide any formal characterisation of their contract languages. This calls for a mathematical foundation of contracts and the formal relationship between clients and contracts.

Along the lines of [6] we describe contracts by a simple CCS-like syntax consisting of just three constructors: sequencing, denoted by a dot, and two infix choice operators $+$ representing the *external choice* (the interacting part decides which one of alternative conversations to carry on); $\oplus$ representing the *internal choice* (the choice is not left to the interacting part). Thus $\alpha.\sigma$ is the contract of services that perform an action α and then implement the contract σ, $\sigma_1 \oplus \sigma_2$ is the contract of services that may decide to implement either σ_1 or σ_2, while $\sigma_1 + \sigma_2$ is the contract of services that according to their client's choice, will implement either σ_1 or σ_2.

Following CCS notation, actions are either write or read actions, the former being topped by a bar, and one being the *co-action* of the other. As a matter of fact, contracts are behavioural types of processes that do not manifest internal moves and the parallel structure. They are *acceptance trees* in Hennessy's terminology [15, 16].

Contracts are then to be used to ensure that interactions between clients and servers will always succeed. Intuitively, this happens if whenever a server offers some set of actions, the client either synchronises with one of them (that is, it performs the corresponding co-action) or it terminates. The server contract will then allow us to determine the set of clients that *comply* with it, that is that will successfully terminate any session of interaction with the server.

A first issue that immediately arises is the one of contract equivalence. For instance it is clear that if a server implements the contract $\overline{a}.(\overline{b} \oplus \overline{c})$, then it also implements $\overline{a}.\overline{b} \oplus \overline{a}.\overline{c}$, and viceversa: writing a and then deciding to write either b or c is equivalent to deciding to write either a and then b or a and then c. So if a client is looking for a service that implements the former contract, then the search engine must return also all services that registered themselves as implementing the latter.

Of course the client will probably be satisfied to interact with services that offer more than what the searched contract specifies: for instance services that propose the client to choose further interactions (e.g. $\sigma_1 + (\overline{a}.(\overline{b} \oplus \overline{c})) + \sigma_2$) or that propose longer interaction patterns (e.g. $\overline{a}.(\overline{b}.c.d \oplus \overline{c})$ would fit the task since every client that successfully terminates with services implementing $\overline{a}.(\overline{b} \oplus \overline{c})$ will also do so with services that implement the latter contracts. This brings a second and more critical issue of defining a subcon-

Proceedings of the 5th ACM SIGPLAN Workshop on Programming Language Technologies for XML (PLAN-X 2007).
January 20, 2007, Nice, France.

tract relation. Intuitively we want to define an order relation on contracts $\sigma_1 \preceq \sigma_2$ such that every client complying with services implementing σ_1 will also comply with services of contract σ_2. For instance, we expect $\overline{a} \oplus \overline{b}.c \preceq \overline{a}$, since every client that terminates with a service that may offer either $\overline{a}$ or $\overline{b}.c$ will also terminate with a service that offers only $\overline{a}$. Similarly, $\overline{a} \preceq \overline{a} + \overline{b}.d$ since a client that terminates with services that implement $\overline{a}$ will also terminate with services that leave the client the choice between $\overline{a}$ and $\overline{b}.d$.

However these last two examples show the main problem of this intuition: it is easy to see that a client that complies with $\overline{a} \oplus \overline{b}.c$ does not necessary comply with $\overline{a} + \overline{b}.d$: if client and server synchronise on b then the client will block trying to write c while the server expects to read d. Therefore under this interpretation $\preceq$ looks as being not transitive:

$$\overline{a} \oplus \overline{b}.c \preceq \overline{a} \ \wedge \ \overline{a} \preceq \overline{a} + \overline{b}.d \quad \not\Longrightarrow \quad \overline{a} \oplus \overline{b}.c \preceq \overline{a} + \overline{b}.d$$

This problem can be solved by resorting to the theory of *explicit coercions* [4]. More in details, the problem of this approach, which is the one proposed in [6], is that servers are used carelessly "as they are". Note indeed that what we are doing here is to use a server of "type" $\overline{a} + \overline{b}.d$ where a server of type $\overline{a} \oplus \overline{b}.c$ is expected. The knowledgeable reader will have recognised that we are using $\preceq$ as an *inverse* subtyping relation for servers[1]. If we denote by $:\!\!>$ the subtyping relation for servers, then $\overline{a} \oplus \overline{b}.c \ :\!\!> \ \overline{a} + \overline{b}.d$ and so what we implicitly did is to apply subsumption [5] and consider that a server that has type $\overline{a} + \overline{b}.d$ has also type $\overline{a} \oplus \overline{b}.c$. The problem is not that $\preceq$ (or, equivalently, $:\!\!>$) is not transitive. The problem is the use of subsumption which corresponds to the use of *implicit* coercions. Coercions are functions that embed objects of a smaller type into a larger type, and that are characterised by the fact that when they are applied to objects of the larger type, their type erasures are the identity function. For instance it is well know that for record types one has $\{a{:}s\} :\!\!> \{a{:}s; b{:}t\}$. This is so because the coercion function $c = \lambda x^{\{a:s;b:t\}}.\{a = x.a\}$ embeds values of the smaller type into the larger one. In order to use a term of type $\{a{:}s; b{:}t\}$ where one of type $\{a{:}s\}$ is expected we first have to embed it in the right type by the coercion function c above, which erases (masks/shields) the b field so that it cannot interfere with the computation. Most programming languages do not require the programmer to write coercions, either because they do not have any actual effect (as in the case of c since the type system already ensures that the b field will never be used) or because they are inserted by the compiler (as when converting an integer into the corresponding float). In this case we speak of *implicit* coercions. However some programming languages (e.g. OCaml) resort to *explicit* coercions because they have a visible effect and, for instance, they cannot be inferred by the compiler.

Coercions for contracts have an observable effect, therefore we develop their meta-theory in term of explicit coercions. However, coercions can be inferred so they can be kept implicit in the language and automatically inserted at static time. Coming back to our example, the embedding of a server of type $\overline{a}$ into $\overline{a} \oplus \overline{b}.c$ is the identity, since we do not have to mask/shield any action of a server of the former type in order to use it in a context where a server of the latter type is expected. On the contrary, to embed a server of type $\overline{a} + \overline{b}.d$ into $\overline{a}$ we have to mask (at least) the $\overline{b}$ action of the server. So in order to use it in a context that expects a $\overline{a}$ server we apply to it a *filter* that will block all $\overline{b}$ messages. Transitivity being a logical cut, the coercion from $\overline{a} + \overline{b}.d$ to $\overline{a} \oplus \overline{b}.c$ is the composition of the two coercions, that is the filter that blocks $\overline{b}$ messages. So if we have a client that complies with $\overline{a} \oplus \overline{b}.c$, then it can be used with a server that implements $\overline{a} + \overline{b}.d$ by applying to this server the filter that blocks its $\overline{b}$ messages. This filter will make the previous problematic synchronisation on b impossible, so the client can do nothing but terminate.

Two observations to conclude this brief overview. First, the fact that we apply filters to servers rather than to clients is just a presentational convenience: the same effect as applying to a server a filter that blocks some actions can be obtained by applying to the client the filter that blocks the corresponding co-actions. Second, filters must be more fine grained in blocking actions than restriction operators as defined for CCS or π. These are "permanent" blocks, while filters are required to be able to modulate blocks along the computation. For instance the filter that embeds $(a.(a + b)) + b.c$ into $a.b$ must block b only at the first step of the reduction and a only at the second step of the reduction.

[1] The inversion is due to the fact that we are considering the client perspective: a contract can be interpreted as the set of clients that comply with services implementing the contract. We decided to keep this notation rather than the inverse one for historical reasons, since it is the same sense as used by De Nicola and Hennessy for the may and must preorders [18].

Overview

In the rest of the work we will formally define our contract language and characterise the subcontract relation in terms of explicit coercion functions (filters). We start by presenting the syntax of our contracts (§2.1), show how to use it to express WSDL and WSCL descriptions (§2.2), and define their semantics (§2.3). We then characterise the set of all clients that are strongly compliant with a service—that is, clients that successfully complete every direct interaction session with the service—and argue that subcontract relations whose definitions are naively based on strong compliance are either too strict or suffer the aforementioned problem of transitivity (§2.4). We argue that subcontracting should not be defined on all possible interactions, but focus only on interactions based on actions that a client expects from the services: all the other possible actions should not interfere with the interaction. We formalise this concept by giving a coinductive definition of a subcontract relation that focuses on this kind of actions, we study its properties and describe the relation with the must preorder and the naively defined relations we introduced (§3.1). The non-interference of unexpected actions is ensured by coercion functions—that we dub *filters*—that, by shielding the actions at issue, embed a service in the "world" of its expected client. We prove that our subcontract relation can be expressed in terms of filters and must preorder and deduce the corresponding weak compliance relation between client and services: a client is weakly compliant with a service if there exists a filter that makes it strongly compliant with it (§3.2) After having studied different characterisations of weak compliance we state the soundness of contracts, namely that a client that is weakly compliant with a service via a given filter will successfully terminate every interaction with the service mediated by the filter (§3.3).

Related work

The two works that are more closely related to this are by Carpineti *et al.* [6] and by Gay and Hole [14]. Carpineti *et al.* is the starting point of our work. There, the syntax of contracts was introduced but subcontracting essentially stopped at the problem of transitivity. In that work compliance is essentially a syntactic notion and the work lacked semantic characterisations of contracts. Gay and Hole's approach is a completely independent work in which session types are introduced for a variant of the π-calculus. Despite the patent differences, this work it is quite close to the work of Carpineti *et al.* insofar as it suffers of the same transitivity problem. The transitivity problem is solved by Gay and Hole at language level, by tailoring the language of processes so as to make impossible the critical case that make transitivity fail. Our approach is more general in that transitivity of subtyping (subcontracting) is directly defined on types, without resorting to a specific process calculus. Our soundness result is stated for a generic process calculus provided that the processes of this calculus can be typed by contracts: this requires lightweight conditions, essentially the satisfaction of subject reduction and the correspondence between the observable behaviour of a process and its contract.

2. CONTRACTS

2.1 Syntax

Let $\mathscr{N}$ be a set of names, we define Σ to be the set of contracts generated by the following grammar.

$$\begin{array}{lcll} \alpha & ::= & a \mid \overline{a} & a \in \mathscr{N} \\ \sigma & ::= & \mathbf{0} \mid \alpha.\sigma \mid \ \sigma \oplus \sigma \mid \sigma + \sigma & \end{array}$$

2.2 Examples

In this section we relate our contract language to existing technologies for specifying service protocols.

2.2.1 *Message exchange patterns in* WSDL

The Web Service Description Language (WSDL) Version 1.1 [10] permits to describe and publish abstract and concrete descriptions of Web services. Such descriptions include the schema [13] of messages exchanged between client and server, the name and type of *operations* that the service exposes, as well as the locations (URLs) where the service can be contacted. In addition, it defines four interaction patterns determining the order and direction of exchanged messages. For instance, the *request-response* pattern is used to describe a synchronous operation where the client issues a request and subsequently receives a response from the service.

The second version of WSDL [3, 8, 9] allows users to agree on message exchange patterns (MEP) by specifying in the required `pattern` attribute of operation elements an absolute URI that identifies the MEP. It is important to notice that these URIs act as global identifiers (their content is not important) for MEPs, whose semantics is usually given in plain English. In particular, WSDL 2.0 [8] predefines four message exchange patterns (each pattern being uniquely identified by a different URI) for describing services where the interaction is initiated by clients. Let us shortly discuss how the informal plain English semantics of these patterns can be formally defined in our contract language. Consider the (simplified) WSDL 2.0 fragment

```
<operation name="A" pattern="in-only">
  <input messageLabel="In"/>
</operation>
<operation name="B" pattern="robust-in-only">
  <input messageLabel="In"/>
  <outfault messageLabel="Fault"/>
</operation>
<operation name="C" pattern="in-out">
  <input messageLabel="In"/>
  <output messageLabel="Out"/>
  <outfault messageLabel="Fault"/>
</operation>
<operation name="D" pattern="in-opt-out">
  <input messageLabel="In"/>
  <output messageLabel="Out"/>
  <outfault messageLabel="Fault"/>
</operation>
```

which defines four operations named `A`, `B`, `C`, and `D`. The first two operations are *asynchronous* by accepting only an incoming message labeled `In`. The last two operations are *synchronous* by accepting an incoming message labeled `In` and replying with a message labeled `Out`. In the `B` operation a fault message can occur after the input. The `C` operation always produces an output message (see `in-out` in its `pattern` attribute), unless a fault occurs. In the `D` operation the reply is optional, as stated by the `in-opt-out` exchange pattern attribute, and again it may fail with `Fault`.

We can encode the contract of the pattern of the `A` operation in our contract language as $\texttt{inOnly} = \texttt{In}.\overline{\texttt{End}}$, that is an input action representing the client's request followed by a message $\overline{\texttt{End}}$ that is sent from the service to notify the client that the interaction has completed.[2]

The `B` operation can be encoded as

$$\texttt{robustInOnly} = \texttt{In}.(\overline{\texttt{End}} \oplus \overline{\texttt{Fault}}.\overline{\texttt{End}})$$

where after the client's request, the interaction may follow two paths, representing successful and faulty computations respectively. In the former case the end of the interaction is immediately signaled to the client. In the latter case a message `Fault` is sent to the client, followed by `End`. The use of the internal choice for combining the two paths states that it is the service that decides whether the interaction is successful or not. This means that a client compliant with this service can either stop after the request or it must be able to handle both the `End` and `Fault` messages: the omission of handling, say, `Fault` would result into an uncaught exception.

The need for an explicit `End` message to signal a terminated interaction is not immediately evident. In principle, the optional fault message could have been encoded as $\texttt{In}.(\mathbf{0} \oplus \overline{\texttt{Fault}})$. A client compliant with this service should be able to receive and handle the `Fault` message, but it must also be able to complete the interaction without further communication from the service. The point is that the client cannot distinguish a completed interaction where the service has internally decided to behave like $\mathbf{0}$ from an interaction where the service has internally decided to behave like `Fault`, but is taking a long time to respond. By providing an explicit `End` message signaling a completed interaction, the service tells the client not to wait for further messages. By

[2]This is just some extra piece of information that the server sends to the client. It signals server's successful termination and it is not related to the success of the whole client/server interaction which is defined by *client*'s termination.

this reasoning, the `End` message after `Fault` is not strictly necessary, but we write it for uniformity.

By similar arguments the contract of the `C` operation can be encoded as

$$\texttt{inOut} = \texttt{In}.(\overline{\texttt{Out}}.\overline{\texttt{End}} \oplus \overline{\texttt{Fault}}.\overline{\texttt{End}})$$

and the contract of the `D` operation as

$$\texttt{inOptOut} = \texttt{In}.(\overline{\texttt{End}} \oplus \overline{\texttt{Out}}.\overline{\texttt{End}} \oplus \overline{\texttt{Fault}}.\overline{\texttt{End}})$$

It is worth noticing that, intuitively, a client that is capable of invoking operation `D` with contract `inOptOut` can also interact successfully with operation `C` with contract `inOut`. Indeed, a client invoking `D` must be able to handle a `End` message, a `Out` message, and a `Fault` message. The `C` operation is "more deterministic" as it can only produce an `Out` message or a `Fault` message. Similarly, `A` is more deterministic than `D` since it can only send an `End` message after the client's request.

2.2.2 *Conversations in* WSCL

The WSDL message exchange patterns cover only the simplest forms of interaction between a client and a service. More involved forms of interactions, in particular stateful interactions, cannot be captured if not as informal annotation within the WSDL interface. The Web service conversation language WSCL [1] provides a more general specification language for describing complex *conversations* between two communicating parties, by means of an activity diagram. The diagram is basically made of *interactions* which are connected with each other by means of *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.

Below we encode the contract σ of a simplified e-commerce service (Figure 1) where the client is required to login before it can issue a query and thus receive 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 was successful, or that the item is out-of-stock, or that the client's payment was refused:

$$\begin{aligned}\sigma \stackrel{\text{def}}{=}\ & \texttt{Login}.(\overline{\texttt{InvalidLogin}}.\overline{\texttt{End}} \oplus \overline{\texttt{ValidLogin}}.\texttt{Query}.\\ & \overline{\texttt{Catalog}}.(\texttt{Logout}.\overline{\texttt{End}} + \texttt{Purchase}.(\\ & \overline{\texttt{Accepted}}.\overline{\texttt{End}} \oplus \overline{\texttt{InvalidPayment}}.\overline{\texttt{End}}\\ & \oplus \overline{\texttt{OutOfStock}}.\overline{\texttt{End}})))\end{aligned}$$

Notice that unlabeled transitions in Figure 1 correspond to external choices in σ, whereas labeled transitions correspond to internal choices. It is also interesting to notice that WSCL explicitly accounts for a termination message (called "empty" in the WSCL specification, the final interaction on the right end in Figure 1) that is used for modeling the end of a conversation. The presence of this termination message finds a natural justification in our formal contract language, as explained above.

Now assume that the service is extended with a booking capability, so that after looking at the catalog the client may book an item to be bought at some later time. The contract of the service would change to σ' as follows:

$$\sigma' \stackrel{\text{def}}{=} \ldots \texttt{Logout}.\overline{\texttt{End}} + \texttt{Book}.\sigma_B + \texttt{Purchase}.(\ldots)$$

It would be desirable for clients that are compliant with the former service to be compliant with this service as well. After all, the extended service offers *more* than the old one. However, assume to have a client that does actually account for a `Book` message from the service and that such a client is compliant with the former service for the simple reason that, since the former service did not provide a booking capability, whatever contract σ'_B the client provided after the `Book` action was irrelevant in order to establish compliance. In the extended service this is no longer the case, and the client can safely interact with the extended service only if the `Book` action is filtered out. This is precisely the transitivity problem we pointed out in the introduction.

2.3 Semantics

Contracts describe the behaviour of the processes that implement them. This behaviour is defined by describing the actions that are offered by a process and the way in which they are offered. This is formally stated by the two definitions given below.

DEFINITION 2.1 (TRANSITION). *Let $\sigma \stackrel{\alpha}{\mapsto\!\!\!\!\!/}$ be the least relation such that:*

$$\begin{array}{rl} \mathbf{0} \stackrel{\alpha}{\mapsto\!\!\!\!\!/} & \\ \beta.\sigma \stackrel{\alpha}{\mapsto\!\!\!\!\!/} & \text{if } \alpha \neq \beta \\ \sigma \oplus \sigma' \stackrel{\alpha}{\mapsto\!\!\!\!\!/} & \text{if } \sigma \stackrel{\alpha}{\mapsto\!\!\!\!\!/} \text{ and } \sigma' \stackrel{\alpha}{\mapsto\!\!\!\!\!/} \\ \sigma + \sigma' \stackrel{\alpha}{\mapsto\!\!\!\!\!/} & \text{if } \sigma \stackrel{\alpha}{\mapsto\!\!\!\!\!/} \text{ and } \sigma' \stackrel{\alpha}{\mapsto\!\!\!\!\!/} \end{array}$$

The transition relation *of contracts, noted $\stackrel{\alpha}{\longmapsto}$, is the least relation satisfying the rules:*

$$\alpha.\sigma \stackrel{\alpha}{\longmapsto} \sigma$$

$$\frac{\sigma_1 \stackrel{\alpha}{\longmapsto} \sigma_1' \quad \sigma_2 \stackrel{\alpha}{\longmapsto} \sigma_2'}{\sigma_1 + \sigma_2 \stackrel{\alpha}{\longmapsto} \sigma_1' \oplus \sigma_2'} \qquad \frac{\sigma_1 \stackrel{\alpha}{\longmapsto} \sigma_1' \quad \sigma_2 \stackrel{\alpha}{\mapsto\!\!\!\!\!/}}{\sigma_1 + \sigma_2 \stackrel{\alpha}{\longmapsto} \sigma_1'}$$

$$\frac{\sigma_1 \stackrel{\alpha}{\longmapsto} \sigma_1' \quad \sigma_2 \stackrel{\alpha}{\longmapsto} \sigma_2'}{\sigma_1 \oplus \sigma_2 \stackrel{\alpha}{\longmapsto} \sigma_1' \oplus \sigma_2'} \qquad \frac{\sigma_1 \stackrel{\alpha}{\longmapsto} \sigma_1' \quad \sigma_2 \stackrel{\alpha}{\mapsto\!\!\!\!\!/}}{\sigma_1 \oplus \sigma_2 \stackrel{\alpha}{\longmapsto} \sigma_1'}$$

and closed under mirror cases for the external and internal choices. We write $\sigma \stackrel{\alpha}{\longmapsto}$ if there exists σ' such that $\sigma \stackrel{\alpha}{\longmapsto} \sigma'$.

The relation $\stackrel{\alpha}{\longmapsto}$ is different from standard transition relations for CCS processes [17]. For example, there is always at most one contract σ' such that $\sigma \stackrel{\alpha}{\longmapsto} \sigma'$, while this is not the case in CCS (the process $a.b + a.c$ has two different a-successor states: b and c). This mismatch is due to the fact that contract transitions define the evolution of conversation protocols *from the perspective of the communicating parties*. Thus $a.b + a.c \stackrel{a}{\longmapsto} b \oplus c$ because, once the action a has been performed, the communicating party is not aware of which conversation path has been chosen. On the contrary, CCS transitions define the evolution of processes *from the perspective of the process itself*.

NOTATION 2.2. *We write $\sigma(\alpha)$ for the unique continuation of σ after α, that is, the contract σ' such that $\sigma \stackrel{\alpha}{\longmapsto} \sigma'$.*

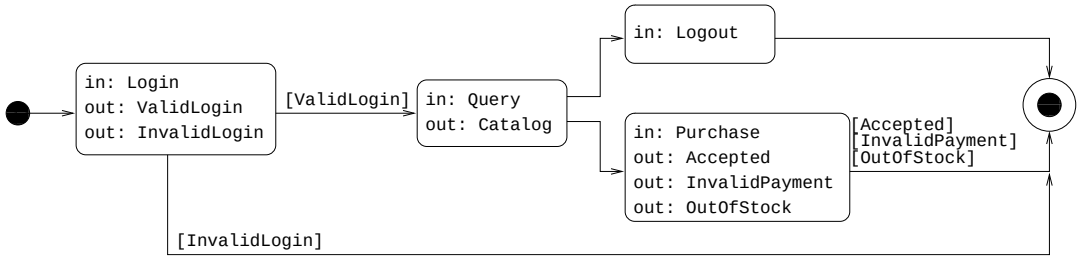

Figure 1: Contract of a simple e-commerce service as a WSCL diagram.

The labelled transition system above describes the actions offered by (a service implementing) a contract, but does not show *how* these actions are offered. In particular the actions offered by an external choice are all available at once while the actions offered by different components of an internal choice are mutually exclusive. Such a description is given by the *ready sets* that are observable for a given contract:

DEFINITION 2.3 (OBSERVABLE READY SETS). *Let $\mathscr{P}_f(\mathcal{N} \cup \overline{\mathcal{N}})$ be the set of finite parts of $\mathcal{N} \cup \overline{\mathcal{N}}$, called* ready sets. *Let also $\sigma \Downarrow \mathrm{R}$ be the least relation between contracts σ in Σ and ready sets R in $\mathscr{P}_f(\mathcal{N} \cup \overline{\mathcal{N}})$ such that:*

$$\begin{array}{ll} \mathbf{0} \Downarrow \emptyset & \\ \alpha.\sigma \Downarrow \{\alpha\} & \\ (\sigma + \sigma') \Downarrow \mathrm{R} \cup \mathrm{R}' & \text{if } \sigma \Downarrow \mathrm{R} \text{ and } \sigma' \Downarrow \mathrm{R}' \\ (\sigma \oplus \sigma') \Downarrow \mathrm{R} & \text{if either } \sigma \Downarrow \mathrm{R} \text{ or } \sigma' \Downarrow \mathrm{R} \end{array}$$

NOTATION 2.4. *We use the convention that the write operation is idempotent, $\overline{\overline{a}} = a$, and for a given ready set R we define its complementary ready set as $\mathtt{co}(\mathrm{R}) = \{\overline{\alpha} \mid \alpha \in \mathrm{R}\}$.*

As a final remark note that in defining the syntax of contracts in Section 2.1 we preferred simplicity over concision. As a matter of fact, this syntax is redundant since the system is equivalent (w.r.t. observable ready sets) to the one in which external choices are always guarded (i.e., they are of the form $\alpha.\sigma + \alpha.\sigma$), since for every observable behaviour of a contract it is possible to define a contract with the same behaviour in which all external choices are guarded.

2.4 The problem

We now possess all the technical instruments to formally state the problem we described in the introduction and recalled at the end of Section 2.2. This first requires the precise definition of *compliance*. Recall that, intuitively, the behaviour of a client complies with the behaviour of a server if for every set of actions that the server may offer, the client either synchronises with one of them, or it terminates. The behaviour of clients, as well as the one of servers, can be described by contracts. Therefore we should define when a contract σ_c (describing the behaviour of a client) complies with another contract σ_s (describing the behaviour of a service). This is done by the following definition

DEFINITION 2.5 (STRONG COMPLIANCE). *The* strong compliance *relation is the largest relation $\dashv_S$ such that $\sigma_c \dashv_S \sigma_s$ implies that:*

1. *for all $\mathrm{R}_c \neq \emptyset$ and R_s such that $\sigma_c \Downarrow \mathrm{R}_c$ and $\sigma_s \Downarrow \mathrm{R}_s$, we have $\mathtt{co}(\mathrm{R}_c) \cap \mathrm{R}_s \neq \emptyset$, and*
2. *for all α, $\sigma_c \overset{\overline{\alpha}}{\longmapsto} \sigma_c'$ and $\sigma_s \overset{\alpha}{\longmapsto} \sigma_s'$ implies $\sigma_c' \dashv_S \sigma_s'$.*

In words the definition above states that a client of contract σ_c is compliant with a server of contract σ_s if, for every possible combination R_s and R_c of the choices of the server and of the client where the client does not terminate immediately ($\mathrm{R}_c \neq \emptyset$), there is an action in the client choice that can synchronise with an action among those offered by the server ($\mathtt{co}(\mathrm{R}_c) \cap \mathrm{R}_s \neq \emptyset$), and the continuation of the client after synchronisation is compliant with the continuation of the server ($\sigma_c' \dashv_S \sigma_s'$).

Once we have such a definition it then looks natural to define the subcontract relation in terms of compliance. A first naive attempt is to define subcontract as:

$$\sigma_1 \preceq_{\text{strong}} \sigma_2 \overset{\text{def}}{\iff} \forall \sigma_c . \sigma_c \dashv_S \sigma_1 \Rightarrow \sigma_c \dashv_S \sigma_2 \qquad (1)$$

Unfortunately this relation is too strong since $\overline{a} \preceq_{\text{strong}} \overline{a} + \overline{b}.d$ does not hold: $a + b.\overline{c}$ is compliant with $\overline{a}$ but not with $\overline{a} + \overline{b}.d$ (actually, $\preceq_{\text{strong}}$ is almost the must preorder: we discuss it in details in Section 3.1.1).

As a second attempt we can try to directly relate two contracts σ_1 and σ_2 on their form, rather than on the sets of their clients. Let $\overline{\sigma}$ denote the complemented contract of σ which, roughly, is obtained by replacing in σ every action by its coaction, every internal choice by an external one, and viceversa (the formal definition is slightly more involved and can be found in [6]). Intuitively $\overline{\sigma}$ denotes the contract of a "canonical" client complying with σ services. Then one can define a new relation on service contracts as:

$$\sigma_1 \ltimes \sigma_2 \overset{\text{def}}{\iff} \overline{\sigma_1} \dashv_S \sigma_2 \qquad (2)$$

This relation—which is the one introduced and studied in [6] —is *nearly* what we are looking for. For instance now we have $\overline{a} \oplus \overline{b}.c \ltimes \overline{a}$ and $\overline{a} \ltimes \overline{a} + \overline{b}.d$, since $\overline{\overline{a} \oplus \overline{b}.c} = a + b.\overline{c} \dashv_S \overline{a}$ and $\overline{\overline{a}} = a \dashv_S \overline{a} + \overline{b}.d$. Unfortunately, $\ltimes$ is not a preorder since transitivity does not hold: $a + b.\overline{c} \not\dashv_S \overline{a} + \overline{b}.d$ implies that $\overline{a} \oplus \overline{b}.c \not\ltimes \overline{a} + \overline{b}.d$.

The reason for such a failure is essentially twofold. First, while contracts account for non-determinism that is internal to each process—being it a client or a server—, they cannot handle the "system" non-determinism that springs from process synchronisation. For instance, the example that made transitivity fail was caused by the interaction between two external choices, $\overline{a} + \overline{b}.d$ and $a + b.\overline{c}$ which yields non-determinism at system level. Second, the approach above seems to suggest that we must look for a subtyping relation for servers such that a server of a given contract "is" compatible with all clients compliant with a subcontract, while in reality what we should be looking for is a relation such

that a server of a given contract "can be made" compatible with all clients compliant with a subcontract. These two points are formally treated in the theory that we expose in the next section.

3. A THEORY OF CONTRACTS

At the end of the previous section we said that we wanted a subcontract relation $\sigma_1 \preceq \sigma_2$ such that a server of contract σ_2 *could be made* compliant with the clients of σ_1 services. The keypoint of the discussion is the "can be made".

Of course we do not want to consider arbitrary transformations of the service, e.g. transformations that alter the semantics of the service. Instead we look for transformations that embed a σ_2 service in a world of σ_1 clients without modifying the behaviour of the service. Therefore we want to shield all actions of the σ_2 that do not belong to the σ_1 world: this means that our shield, when applied to a σ_1 service, has absolutely no effect. This is precisely the definition of an explicit coercion from σ_2 to σ_1 (recall that the subcontract relation is the inverse of a service subtyping relation; *c.f.* Footnote 1): an embedding function that behaves as the identity on the co-domain.

3.1 Weak subcontract relation

The idea is that $\sigma_1 \preceq \sigma_2$ if there exists some (possibly empty) set of actions belonging only to the world of σ_2 that, if shielded, can make a σ_2 service behave as a σ_1 service. This is formally stated by the following definition:

DEFINITION 3.1 (WEAK SUBCONTRACT). *The* weak subcontract relation $\preceq$ *is the largest relation such that* $\sigma_1 \preceq \sigma_2$ *implies that*

1. *for all* R *such that* $\sigma_2 \Downarrow \mathrm{R}$ *there exists* $\mathrm{S_R} \subseteq \mathrm{R}$ *such that* $\sigma_1 \Downarrow \mathrm{S_R}$ *and*
2. *for all* $\alpha \in \mathrm{S_R}$ *we have* $\sigma_1(\alpha) \preceq \sigma_2(\alpha)$.

The basic intuition about the weak subcontract relation is that a client that interacts successfully with a service with contract σ_1 must be able to complete whatever ready set $\mathrm{R_1}$ is chosen from σ_1. So, if we want to replace the service with another one whose contract is σ_2, we require that whatever ready set $\mathrm{R_2}$ is chosen from σ_2 there is a smaller one $\mathrm{R_1} \subseteq \mathrm{R_2}$ in σ_1 such that all of the continuations with respect to the actions in $\mathrm{R_1}$ are in the weak subcontract relation. However, in order to avoid interferences we might need to filter out the actions in $\mathrm{R_2} \setminus \mathrm{R_1}$.

PROPOSITION 3.2. $\preceq$ *is a precongruence with respect to* $+$ *and* $\oplus$ *and the prefix operators.*

3.1.1 Comparison with other relations

How does our subcontract relation $\preceq$ compare with existing relations and those introduced in Section 2.4? In Section 2.4 we said that the relation $\ltimes$ defined by equation (2) was nearly what we sought for, but for the lack of transitivity it was not a preorder. The following theorem shows that $\preceq$ obviates this problem:

THEOREM 3.3. *The subcontract relation* $\preceq$ *is the transitive closure of* $\ltimes$.

PROOF. *Appendix A.*

A second relation that one may want to consider is the relation $\preceq_{\text{strong}}$ defined in Section 2.4 by equation (1). It is not difficult to see that this relation is contained in $\preceq$. The definition of $\preceq_{\text{strong}}$ is nice because it derives from strong compliance but we know that it is too strong for our aims. Furthermore $\preceq_{\text{strong}}$ is not a pre-congruence with respect to $+$, this means that $\preceq_{\text{strong}}$ is too weak to be used for safe substitutability of processes in the subsumption rule. Therefore we define yet a different relation which, despite being slightly stronger than $\preceq_{\text{strong}}$, has better properties and is well studied:

DEFINITION 3.4 (STRONG SUBCONTRACT). *The* strong subcontract *relation is the largest relation* $\sqsubseteq$ *such that* $\sigma_1 \sqsubseteq \sigma_2$ *implies that*

1. *for all* $\mathrm{R_2}$ *such that* $\sigma_2 \Downarrow \mathrm{R_2}$ *there exists* $\mathrm{R_1} \subseteq \mathrm{R_2}$ *such that* $\sigma_1 \Downarrow \mathrm{R_1}$*, and*
2. *for all* α *we have that* $\sigma_2 \stackrel{\alpha}{\longmapsto} \sigma_2'$ *implies there exists* σ_1' *such that* $\sigma_1 \stackrel{\alpha}{\longmapsto} \sigma_1'$ *and* $\sigma_1' \sqsubseteq \sigma_2'$.

We write $\sigma_1 \simeq \sigma_2$ *if* $\sigma_1 \sqsubseteq \sigma_2$ *and* $\sigma_2 \sqsubseteq \sigma_1$.

The reader might have recognised that the strong subcontract relation is nothing but the *must preorder*. This relation is a bit stronger than $\preceq_{\text{strong}}$[3] (for example $\mathbf{0} \preceq_{\text{strong}} \sigma$ always holds, whereas in general $\mathbf{0} \sqsubseteq \sigma$ does not), but it is also more stable with respect to the type constructors and the compliance relation. This is shown by the following theorem which states that it is safe to replace a service contract with another one that is more deterministic, still preserving the strong compliance property.

THEOREM 3.5. *If* $\sigma_c \dashv_S \sigma_s$ *and* $\sigma_s \sqsubseteq \sigma_s'$ *then* $\sigma_c \dashv_S \sigma_s'$.

PROOF. For condition 1 of strong compliance, let S' be a ready set of σ_s' and R be a nonempty ready set of σ_c. We have, by definition of $\sqsubseteq$, that there exists a ready set S of σ_s which is included in S', and then we know by definition of $\dashv_S$ that this set has a nonempty intersection with $\mathtt{co}(\mathrm{R})$. Then also: $\mathtt{co}(\mathrm{R}) \cap \mathrm{S}' \neq \emptyset$.

For condition 2, let α be such that $\sigma_s' \stackrel{\alpha}{\longmapsto}$ and $\sigma_c \stackrel{\overline{\alpha}}{\longmapsto}$, then we also have $\sigma_s \stackrel{\alpha}{\longmapsto}$ and $\sigma_s(\alpha) \sqsubseteq \sigma_s'(\alpha)$ by definition of $\sqsubseteq$, and then $\sigma_c(\overline{\alpha}) \dashv_S \sigma_s(\alpha)$ by definition of $\dashv_S$. So we can just do an induction on the depth of σ_c. □

The $\sqsubseteq$ is also nicer than $\preceq_{\text{strong}}$ because its very definition shows the differences with $\preceq$. If we compare Definition 3.1 and Definition 3.4, we see that they differ on the set of α considered in condition 2. The latter requires that whatever interaction may happen between a client and a server, the relation must be satisfied by the continuations. The former instead requires this to happen only for interactions on actions that are expected for the smaller contract. This means

[3]The difference is the fact that as soon as two server contracts have an empty ready set, they become indistinguishable in terms of compliance: the only client which complies with them is the null client, and thus they are equivalent in terms of $\preceq_{\text{strong}}$. This is also what makes the precongruence property fail for $\preceq_{\text{strong}}$ w.r.t. the $+$ operation. Indeed, the external choice does not preserve the existence of an empty ready set ($\sigma_1 + (\mathbf{0} \oplus \sigma_2)$ is equivalent to $\sigma_1 \oplus (\sigma_1 + \sigma_2)$), and thus not being able to distinguish between $\mathbf{0}$ and $\mathbf{0} \oplus \sigma_2$ leads us into trouble. So we see that the extra sub-classification the must preorder does in the big class of contracts having an empty ready set is necessary to have a precongruence.

that with the weak subcontract relation all the actions that are not expected by the smaller contract *must not* take part in the client-server interaction. If we want to replace a server by a different server with a (weak) super-contract, then we must ensure that the client is shielded from these unexpected actions. The technical instrument to ensure it are the *filters* we define next.

3.2 Filters

A filter is simply the specification of a set of actions that are allowed at a certain time, along with the continuation filters after that action has occurred:

$$f ::= \coprod_{\alpha \in \mathrm{A}} \alpha.f_\alpha$$

By convention we use $\mathbf{0}$ for denoting the empty filter, that is the filter that allows no action ($\mathrm{A} = \emptyset$). Filters have a simple transition relation, as follows:

$$\coprod_{\alpha \in \mathrm{A}} \alpha.f_\alpha \overset{\beta}{\longmapsto} f_\beta \qquad \text{if } \beta \in \mathrm{A}$$

As usual we write $f \overset{\alpha}{\not\longmapsto}$ if there is no f' such that $f \overset{\alpha}{\longmapsto} f'$. The application of a filter f to a contract σ, written $f(\sigma)$, produces another contract where only the allowed actions are visible:

$$\begin{array}{rcll} f(\mathbf{0}) & = & \mathbf{0} & \\ f(\alpha.\sigma) & = & \mathbf{0} & \text{if } f \overset{\alpha}{\not\longmapsto} \\ f(\alpha.\sigma) & = & \alpha.f'(\sigma) & \text{if } f \overset{\alpha}{\longmapsto} f' \\ f(\sigma_1 + \sigma_2) & = & f(\sigma_1) + f(\sigma_2) & \\ f(\sigma_1 \oplus \sigma_2) & = & f(\sigma_1) \oplus f(\sigma_2) & \end{array}$$

Filters allow us to express the weak subcontract relation in terms of strong one:

THEOREM 3.6. *$\sigma_1 \preceq \sigma_2$ if and only if there exists a filter f such that $\sigma_1 \sqsubseteq f(\sigma_2)$.*

PROOF. The proof is interesting since it shows the exact relation between filters and the $\mathrm{S_R}$ of Definition 3.1. The proof is by induction on the depth of σ_1.

($\Rightarrow$) If the depth of σ_1 is 0, then it is sufficient to take $f = \mathbf{0}$ (the empty filter that allows no action). If the depth of σ_1 is $n > 0$, then assume that the theorem holds for every σ_1 with depth smaller than n. We know that $\sigma_2 \Downarrow \mathrm{R}$ implies that there exists $\mathrm{S_R} \subseteq \mathrm{R}$ such that $\sigma_1 \Downarrow \mathrm{S_R}$ and for all $\alpha \in \mathrm{S_R}$ we have $\sigma_1(\alpha) \preceq \sigma_2(\alpha)$. By induction hypothesis we know that for every $\alpha \in \mathrm{S_R}$ there exists f'_α such that $\sigma_1(\alpha) \sqsubseteq f'_\alpha(\sigma_2(\alpha))$. Let us define

$$S \overset{\text{def}}{=} \bigcup_{\sigma_2 \Downarrow \mathrm{R}} \mathrm{S_R} \qquad \text{and} \qquad f \overset{\text{def}}{=} \coprod_{\alpha \in S} \alpha.f'_\alpha$$

We have to show that $\sigma_1 \sqsubseteq f(\sigma_2)$. Assume $\sigma_2 \Downarrow \mathrm{R}$. Then $f(\sigma_2) \Downarrow \mathrm{R} \cap S$. Since S is the union of all the $\mathrm{S_R}$, we know that $\mathrm{S_R} \subseteq \mathrm{R} \cap S$. Now assume that $f(\sigma_2) \overset{\alpha}{\longmapsto} f'_\alpha(\sigma_2(\alpha))$ then $\alpha \in S$ and hence $\alpha \in \mathrm{S_R}$ for some $\mathrm{S_R}$ such that $\sigma_1 \Downarrow \mathrm{S_R}$. Thus we have that $\sigma_1 \overset{\alpha}{\longmapsto} \sigma_1(\alpha)$ and, by induction hypothesis, $\sigma_1(\alpha) \sqsubseteq f'_\alpha(\sigma_2(\alpha))$, from which we conclude $\sigma_1 \sqsubseteq f(\sigma_2)$.

($\Leftarrow$) If the depth of σ_1 is 0, then the theorem follows immediately since σ_1 is the smallest contract. If the depth of σ_1 is $n > 0$, then assume that the theorem holds for every σ_1 with depth smaller than n. From $\sigma_1 \sqsubseteq f(\sigma_2)$ we know that $f(\sigma_2) \Downarrow \mathrm{R}$ implies that there exists $\mathrm{R}' \subseteq \mathrm{R}$ such that $\sigma_1 \Downarrow \mathrm{R}'$. By definition of $f(\sigma_2)$ we have $\sigma_2 \Downarrow \mathrm{R}''$ with $\mathrm{R} \subseteq \mathrm{R}''$ and now $\mathrm{R}' \subseteq \mathrm{R} \subseteq \mathrm{R}''$. Now let us take $\alpha \in \mathrm{R}'$. Since $\alpha \in \mathrm{R}$ we have $f(\sigma_2) \overset{\alpha}{\longmapsto} f'(\sigma_2(\alpha))$ and $\sigma_1 \overset{\alpha}{\longmapsto} \sigma_1(\alpha)$ and $\sigma_1(\alpha) \sqsubseteq f'(\sigma_2(\alpha))$ for some filter f'. By induction hypothesis we have $\sigma_1(\alpha) \preceq \sigma_2(\alpha)$ from which we conclude $\sigma_1 \preceq \sigma_2$. □

Let us consider again our example of $\overline{a} \oplus \overline{b}.c$ and $\overline{a} + \overline{b}.d$. These contracts are not related by the strong subcontract relation, but any client complying with the first one has to be ready to read on a and then terminate. Then, we see that the second one *can be made* compliant with any such client, because it is ready to write on a: so we are sure that synchronisation on a is possible, and that if it occurs the client will terminate. The point is then to ensure that this synchronisation will indeed occur and that the channel b will not be selected instead, which would lead to deadlock. This is done by applying to $\overline{a} + \overline{b}.d$ the filter $f = \overline{a}.\mathbf{0}$, which lets the sole action $\overline{a}$ pass. Formally, we have that $f(\overline{a} + \overline{b}.d) = \overline{a}$, and $\overline{a} \oplus \overline{b}.c \sqsubseteq \overline{a}$ holds.

Filters can also be used as proofs (in the sense of the Curry-Howard isomorphism) for the weak subcontract relation. First we have to define the "identity" filter, that is the one that proves isomorphic (w.r.t. filter morphisms) contracts.

DEFINITION 3.7. *The* identity filter *for a contract σ, notation I_σ, is defined as*

$$I_\sigma \overset{\text{def}}{=} \coprod_{\sigma \overset{\alpha}{\longmapsto} \sigma'} \alpha.I_{\sigma'}$$

It is easy to see that $I_\sigma(\sigma) = \sigma$.

Then the weak subcontract relation can be axiomatised as stated in Table 1, where $\mathtt{init}(\sigma) = \{\alpha \mid \sigma \overset{\alpha}{\longmapsto}\}$ and we use $\sigma_1 = \sigma_2$ as a shorthand for $I_{\sigma_1} : \sigma_1 \leq \sigma_2 \ \wedge \ I_{\sigma_2} : \sigma_2 \leq \sigma_1$.

LEMMA 3.8.
a. If $f : \sigma_1 \leq \sigma_2$, then $\sigma_1 \sqsubseteq f(\sigma_2)$,
b. If $\sigma_1 \sqsubseteq f'(\sigma_2)$, then there exists f such that $f : \sigma_1 \leq \sigma_2$

By combining Theorem 3.6 and Lemma 3.8 we obtain:

COROLLARY 3.9. *$\sigma_1 \preceq \sigma_2$ if and only if there exists a filter f such that $f : \sigma_1 \leq \sigma_2$.*

3.2.1 Weak compliance

As we defined a strong compliance relation, so we can define a weak compliance relation in which interactions are limited to actions belonging to the "client" world:

DEFINITION 3.10 (WEAK COMPLIANCE). *The* weak compliance *relation is the largest relation $\dashv_W$ such that whenever $\sigma_c \dashv_W \sigma_s$ holds, there exists a set of actions $\mathrm{A} \subseteq \mathscr{N} \cup \overline{\mathscr{N}}$ such that:*

1. *for all $\mathrm{R_c} \neq \emptyset$ and R_s such that $\sigma_c \Downarrow \mathrm{R_c}$ and $\sigma_s \Downarrow \mathrm{R}_s$, we have $\mathrm{A} \cap \mathtt{co}(\mathrm{R_c}) \cap \mathrm{R}_s \neq \emptyset$, and*
2. *for any α in A, $\sigma_c \overset{\overline{\alpha}}{\longmapsto} \sigma'_c$ and $\sigma_s \overset{\alpha}{\longmapsto} \sigma'_s$ implies $\sigma'_c \dashv_W \sigma'_s$.*

Note how the existence of the ready set A in the above definition corresponds to the existence of $\mathrm{S_R}$ in Definition 3.1

As we did for the weak subcontract relation, the weak compliance relation can be decomposed in terms of filters and strong relation:

THEOREM 3.11.

$$\sigma_c \dashv_W \sigma_s \iff \exists \sigma \preceq \sigma_s . \sigma_c \dashv_S \sigma \qquad (3)$$
$$\iff \exists f . \sigma_c \dashv_S f(\sigma_s) \qquad (4)$$

Table 1: Axiomatisation of the weak subcontract relation.

$$\begin{array}{rl} \sigma + \sigma = \sigma & \sigma \oplus \sigma = \sigma \\ \sigma + \sigma' = \sigma' + \sigma & \sigma \oplus \sigma' = \sigma' \oplus \sigma \\ \sigma + (\sigma' + \sigma'') = (\sigma + \sigma') + \sigma'' & \sigma \oplus (\sigma' \oplus \sigma'') = (\sigma \oplus \sigma') \oplus \sigma'' \\ \sigma + (\sigma' \oplus \sigma'') = (\sigma + \sigma') \oplus (\sigma + \sigma'') & \sigma \oplus (\sigma' + \sigma'') = (\sigma \oplus \sigma') + (\sigma \oplus \sigma'') \end{array}$$

$$\sigma + \mathbf{0} = \sigma \qquad \alpha.\sigma + \alpha.\sigma' = \alpha.(\sigma \oplus \sigma') \qquad \alpha.\sigma \oplus \alpha.\sigma' = \alpha.(\sigma \oplus \sigma')$$

$$I_{\sigma \oplus \sigma'} : \sigma \oplus \sigma' \leq \sigma$$

(Service-Ext) $\dfrac{\mathtt{init}(\sigma) \cap \mathtt{init}(\sigma') = \emptyset}{I_\sigma : \sigma \leq_s \sigma + \sigma'}$ (C-Prefix) $\dfrac{f : \sigma \leq \sigma'}{\alpha.f : \alpha.\sigma \leq \alpha.\sigma'}$ (C-IChoice) $\dfrac{f : \sigma \leq \sigma'}{f \vee I_{\sigma''} : \sigma \oplus \sigma'' \leq \sigma' \oplus \sigma''}$ (Trans) $\dfrac{f : \sigma \leq \sigma' \qquad f' : \sigma' \leq \sigma''}{f \wedge f' : \sigma \leq \sigma''}$

Table 2: Meta-operations on filters.

$$\begin{array}{rcl} \coprod_{\alpha \in A} \alpha.f_\alpha \wedge \coprod_{\alpha \in B} \alpha.g_\alpha & \stackrel{\text{def}}{=} & \coprod_{\alpha \in A \cap B} \alpha.(f_\alpha \wedge g_\alpha) \\ \coprod_{\alpha \in A} \alpha.f_\alpha \vee \coprod_{\alpha \in B} \alpha.g_\alpha & \stackrel{\text{def}}{=} & \coprod_{\alpha \in A \cap B} \alpha.(f_\alpha \vee g_\alpha) \coprod_{\alpha \in A \setminus B} \alpha.f_\alpha \coprod_{\alpha \in B \setminus A} \alpha.g_\alpha \end{array}$$

3.2.2 Set-theoretic interpretation

In Section 2.4 we defined a strong subcontract relation, $\preceq_{\text{strong}}$, in terms of the strong compliance, by saying that a contract is a subcontract of another if every contract that complies with the former also complies with the latter.

This corresponds to a set-theoretic interpretation of a contract as the set of clients that comply with it. The subcontract relation coincides with set inclusion. The fact of having such an interpretation and the correspondence between inclusion and subcontracting is an important feature that permits the use of semantic subtyping and paves the way to use of powerful set-theoretic search primitives based on intersection, union, and negation [7].

Now that we have defined a weak compliance relation and a weak subcontract relation, we will show that there is a similar set-theoretic interpretation of the weak subcontract relation in terms of weak compliance.

Theorem 3.12. *$\sigma_1 \preceq \sigma_2$ if and only if for all σ, $\sigma \dashv_W \sigma_1$ implies $\sigma \dashv_W \sigma_2$.*

Proof. *Appendix A.*

Remark 3.13 (Client subcontract relation). *The interpretation of the subcontract relation as the subset relation on the sets of clients complying with the contracts suggests to consider a dual interpretation for clients. We can define the relation $\sigma_1 \preceq_c \sigma_2$ as*

$$\sigma_1 \preceq_c \sigma_2 \stackrel{def}{\iff} \forall \sigma_s . \sigma_1 \dashv_W \sigma_s \Rightarrow \sigma_2 \dashv_W \sigma_s \qquad (5)$$

The above definition is not very informative. For that it is better to consider a coinductive characterisation of $\preceq_c$, as the largest relation such that $\sigma_1 \preceq_c \sigma_2$ implies:

$$\forall \mathrm{R}_2 \neq \emptyset, \sigma_2 \Downarrow \mathrm{R}_2 \Rightarrow \exists \mathrm{R}_1 \neq \emptyset, \begin{cases} \sigma_1 \Downarrow \mathrm{R}_1 \text{ and } \mathrm{R}_1 \subseteq \mathrm{R}_2 \text{ and} \\ \forall \alpha \in \mathrm{R}_1, \sigma_1(\alpha) \preceq_c \sigma_2(\alpha) \end{cases}$$

This definition is equivalent to (5) (the proof is very similar to the one of Theorem 3.12). The only difference between this definition and Definition 3.1 is that in the former only nonempty ready sets are considered. This reflects exactly the asymmetry of the compliance relations (weak and strong) where the empty ready set of the client, if it exists, plays no role.

It is worth mentioning that despite its definition being very close to the weak server subcontract relation, this client subcontract relation has some surprising properties: for example, it is not even sufficient that $\sigma_1 \sqsubseteq \sigma_2$ to get $\sigma_1 \preceq_c \sigma_2$.

Finally filters have an operational meaning, since they allow us to state the soundness of our type system. This can be roughly expressed as the fact that given a server and a weakly compliant client, every interaction between them mediated by the filter that proves the weak compliance (Theorem 3.11(2)) will be successful (the client terminates). This is formally stated in the following section.

3.3 Language

In this contribution we do not consider any particular process language. Instead, we assume that such a language is equipped with a labelled transition system so that

$$P \xrightarrow{\mu} P'$$

describes the evolution of a process P that performs a μ action thus becoming the process P'. Here, μ can either be

a visible action of the form a or $\overline{a}$, or it can be an internal action τ. As usual, we write α for a generic visible action.

DEFINITION 3.14. *Let $P \parallel Q \longrightarrow P' \parallel Q'$ be the least relation such that:*

- *if $P \xrightarrow{\tau} P'$ then $P \parallel Q \longrightarrow P' \parallel Q$;*
- *if $Q \xrightarrow{\tau} Q'$ then $P \parallel Q \longrightarrow P \parallel Q'$;*
- *if $P \xrightarrow{\alpha} P'$ and $Q \xrightarrow{\overline{\alpha}} Q'$ then $P \parallel Q \longrightarrow P' \parallel Q'$.*

Let $P \dashv_S Q$, read P is strongly compliant *with Q, if one of the following holds:*

1. *$P \xrightarrow{\mu}\!\!\!\!\!\!/\;$ for every μ, or*
2. *$P \parallel Q \longrightarrow P' \parallel Q'$ and $P' \dashv_S Q'$.*

The intuition of this definition is that $P \parallel Q$ represents a client P and a service Q interacting with each other. When $P \dashv_S Q$ then every interaction between P and Q terminates with P being reduced to a process equivalent to the null process (P cannot emit any α) which denotes successful termination for P.

We also assume that a type system is given to extract the contract from a process, where

$$\vdash P : \sigma$$

means that in such type system it is possible to derive that the process P has contract σ. Of course the typing and reduction relations must satisfy some basic properties: essentially, contracts must describe the observational behaviour of processes and the reduction must reduce non-determinism (entropy must always increase). This requires the following auxiliary definition.

DEFINITION 3.15. *The relation $\sigma \longrightarrow \sigma'$, read σ' is* more deterministic *than σ, is the least relation satisfying the rules*

$$\sigma_1 \oplus \sigma_2 \longrightarrow \sigma_1 \qquad \frac{\sigma_1 \longrightarrow \sigma_1'}{\sigma_1 + \sigma_2 \longrightarrow \sigma_1' + \sigma_2}$$

and closed under mirror cases for the external and internal choices. As usual, we write $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$.

The type system must obey the following consistency conditions: if $\vdash P : \sigma$ and $P \xrightarrow{\mu} P'$ then $\vdash P' : \sigma'$ and

1. if $\mathbf{0} \sqsubseteq \sigma$, then $\mu = \tau$;
2. if $\mu = \tau$, then $\sigma \Longrightarrow \sigma'$;
3. if $\mu = \alpha$, then $\sigma \xmapsto{\alpha} \Longrightarrow \sigma'$.

Intuitively, condition (1) states that if a process has a null contract, then it can only perform internal actions. Condition (2) states that a process performing internal actions can only make its contract more deterministic. Condition (3) states that if a process performs a visible action α, then its contract must account for that action and the contract of the resulting process P' is (more deterministic than) the contract $\sigma(\alpha)$, which accounts for all the possible behaviours of P after α.[4]

The following lemma is the counterpart of Theorem 3.5 and it states that it is possible to replace a client contract σ_c with another one which is more deterministic, still preserving the compliance property. The lemma is fundamental in proving the soundness of the type system.

LEMMA 3.16. *If $\sigma_c \dashv_S \sigma_s$ and $\sigma_c \Longrightarrow \sigma_c'$ then $\sigma_c' \dashv_S \sigma_s$.*

PROOF. The proof is trivial since the ready sets of σ_c' are a subset of the ready sets of σ_c. □

The soundness of the type system is ensured by the following result, stating that if the contracts of two processes comply, the corresponding processes comply as well, guaranteeing termination on the client side.

THEOREM 3.17. *If $\vdash P : \sigma$ and $\vdash Q : \sigma'$ and $\sigma \dashv_S \sigma'$ then $P \dashv_S Q$.*

PROOF. *Appendix A.*

Notice that the soundness theorem holds when the client's contract and the service's contract are strongly compliant. To be able to use a service for which we only have a weakly compliant client, we need to shield potentially dangerous service actions by means of a filter. Thus, we enrich the process language with a construct

$$f[P]$$

that applies a filter f to a process P, the idea being that the filter constraints the set of visible actions of P, that is its capabilities to interact with the environment, still not altering its behaviour. The labelled transition system of the language is consequently enriched with the following two inference rules:

$$\text{(FILTER1)}\ \frac{P \xrightarrow{\alpha} P' \qquad f \xmapsto{\alpha} f'}{f[P] \xrightarrow{\alpha} f'[P']} \qquad \text{(FILTER2)}\ \frac{P \xrightarrow{\tau} P'}{f[P] \xrightarrow{\tau} f[P']}$$

The introduction of filters into process has consequences on the type system as well. Since our discussion is parametric in the process language and in the type system, we only need to show that the typing judgement

$$\text{(T-FILTER)}\ \frac{\vdash P : \sigma}{\vdash f[P] : f(\sigma)}$$

is consistent in the sense that the three conditions above still hold in the process language extended with filters. Condition (1) is trivially satisfied, since if $\mathbf{0} \sqsubseteq f(\sigma)$ holds there are two possibilities: either $\mathbf{0} \sqsubseteq \sigma$ in which case P has no visible actions and so does $f[P]$, or it is f that masks all of the visible actions of P, in which case they are forbidden by (FILTER1). As regards condition (2) this may happen only if $P \xrightarrow{\tau} P'$ and is then an immediate consequence that $\sigma \Longrightarrow \sigma'$ implies $f(\sigma) \Longrightarrow f(\sigma')$. Finally, as regards condition (3), assume that $P \xrightarrow{\alpha} P'$ and $\vdash P' : \sigma'$. There are two

[4] A reasonable (i.e. informative) type system would also satisfy the condition that if $\vdash P : \sigma$, then for every $\alpha \in \mathtt{init}(\sigma)$ there exists P' such that $P \xrightarrow{\tau}{}^* \xrightarrow{\alpha} P'$. That is, the type system does not deduce capabilities that a process does not have. This however is not necessary for our soundness result.

possibilities: if $f \stackrel{\alpha}{\not\longmapsto}$, then $f[P] \stackrel{\alpha}{\not\longrightarrow}$ and there is nothing to prove. If $f \stackrel{\alpha}{\longmapsto} f'$, then $\sigma(\alpha) \Longrightarrow \sigma'$. Now

$$f(\sigma)(\alpha) = f'(\sigma(\alpha)) \Longrightarrow f'(\sigma')$$

again since $\sigma \Longrightarrow \sigma'$ implies $f(\sigma) \Longrightarrow f(\sigma')$.

The following result summarises the contribution of our work: the adoption of filters enlarges the number of possible services that can be used to let a client terminate.

COROLLARY 3.18. *If* $\vdash P : \sigma_c$, $\vdash Q : \sigma_s$, *and* $\sigma_c \dashv_S f(\sigma_s)$, *then* $P \dashv_S f[Q]$.

As an aside it is nice to notice that filters can encode CCS and π-calculus restrictions: $(\nu a)P = f_{aP}[P]$ where

$$f_{aP} = \coprod_{\alpha \in (\mathrm{fn}(P) \cup \mathsf{co}(\mathrm{fn}(P))\setminus\{a,\overline{a}\}} \alpha.f_{aP}$$

Note that in our case since our contracts are finite we can unfold the filter above to the depth of P and obtain a finite filter.

4. CONCLUSION AND FUTURE WORK

This paper provides a foundation for behavioural typing of web services and it promotes service reuse and/or redefinition by the introduction of a subcontract relation. This work improves the one by Carpineti *et al.* [6] in a number of ways: first of all, we give a direct characterisation of (strong) compliance between a client and a service, and we use this notion to develop a sound and meaningful subcontract relation with interesting properties (in particular, it is a pre-congruence). In Carpineti *et al.*, the subcontract relation $\ltimes$ was defined first, and compliance was obtained by means of a delicate dual operator over contracts. Second, the subcontract relation $\preceq$ as defined in this paper is the transitive closure of $\ltimes$, and we have been able to provide both co-inductive and set-theoretic definitions for it. Furthermore, we claim that the axiomatisation given in Table 1 is sound and complete. Third, as an interesting by-product of the new subcontract relation we have proposed filters as an original programming construct. Filters are necessary for guaranteeing successful interaction between a client and a service when the respective contracts are only weakly compliant, whereas in Carpineti *et al.* only strong compliance was taken into account. As we showed at the end of Section 3, filters can encode CCS and π-calculus restrictions. However it is not clear whether filters are more expressive than restriction, and this issue is worth of future work.

This work also improves the literature on session types on the particular aspect of language neutrality. Even though many session type systems are more expressive than ours (e.g. by using recursive types and allowing name passing), they all avoid the central problem we tackled in this work, that is the system-wide non-determinism involved in the resolution of choices which are external to both client and server. This is usually (e.g. in [14]) resolved at language level, by forcing to have different constructions for internal and external choices and allowing the former to synchronise only with the latter and vice-versa. Here instead we have a more liberal approach, since we do not require the language to implement the different choices but just to emit some signals: it is the way in which these signals are offered that determines whether they form an internal or an external choice.

Even if in this presentation we applied filters to services, in practice it is the client's responsibility to apply them. A client searching for a service with a given contract will receive as answer to its query the reference of a service together with a filter that will allow the client to use the service. Thus the filter must be computed by the query engine. This is not possible yet: as stated in the title this still is work in progress, and the study lacks all the algorithmic part. The axiomatisation of the weak subcontract relation is not operational, we do not have a way to derive a canonical filter for a given subcontract and derive the coherence of the system. Actually we do not even have the definition of canonical filter: if we had it then we could have stated Lemma 3.8 as an "if and only if" instead of splitting it into two separate statements. Of course this is one of the challenges we shall tackle next.

Several other future research directions stem from this work. Here is a non-exhaustive list:

Polarised contracts On the lines of [14] we can study a restricted version of the contract language in which system wide non-determinism is made impossible by polarising the choices operators so that external choices are always guarded by read actions and internal choices are guarded by write actions.

Asymmetric choices The choice operators are commutative. We could try to relax this property in order to give the summands different priorities, which is impossible with the current definitions. For instance there is no way for a client that has to use a service with contract $(a+b) \oplus a$ to specify that it wants to connect with b if this action is available, and with a otherwise.

Contract isomorphisms The only morphisms between contracts we have considered are filters. Since filters are coercions, then by definition they essentially do not alter the semantics of objects. One could try to consider more expressive morphisms (e.g. renaming and/or reordering of actions) and to completely characterise the isomorphisms of contracts. This would allow us to perform service discovery modulo isomorphisms: when searching for services of a given contract a client could be returned a service and two conversion functions, one to call the service, the other to convert results (see [19, 12]).

This could later be extended to richer query/discovery languages obtained by adding union, intersection and negation types on the basis of the set-theoretic interpretation presented here and the work on semantic subtyping [7].

Recursion and higher order The contracts and filters we discussed in this work are finite. An obvious extension to consider is the introduction of recursion both in contracts and, consequently, in filters. Also, for the time being synchronisation does not carry any information. Thus a further step would be to introduce higher order channels *à la* π-calculus.

Relation with other formalisms Finally, connection with other formalisms such as linear logic, session types, or game semantics must surely be investigated more deeply.

5. REFERENCES

[1] 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`.

[2] 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`.

[3] David Booth and Canyang Kevin Liu. *Web Services Description Language (*WSDL*) Version 2.0 Part 0: Primer*, March 2006. `http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327`.

[4] K. Bruce and G. Longo. A modest model of records, inheritance and bounded quantification. *Information and Computation*, 87(1/2):196–240, 1990.

[5] L. Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76:138–164, 1988. A previous version can be found in Semantics of Data Types, LNCS 173, 51-67, Springer, 1984.

[6] 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.

[7] G. Castagna and A. Frisch. A gentle introduction to semantic subtyping. In Proceedings of *PPDP '05, the 7th ACM SIGPLAN International Symposium on Principles and Practice of Declarative Programming,* ACM Press (full version) and *ICALP '05, 32nd International Colloquium on Automata, Languages and Programming,* Lecture Notes in Computer Science n. 3580, Springer (summary), Lisboa, Portugal, 2005. Joint ICALP-PPDP keynote talk.

[8] Roberto Chinnici, Hugo Haas, Amelia A. Lewis, Jean-Jacques Moreau, et al. *Web Services Description Language (*WSDL*) Version 2.0 Part 2: Adjuncts*, March 2006. `http://www.w3.org/TR/2006/CR-wsdl20-adjuncts-20060327`.

[9] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana. *Web Services Description Language (*WSDL*) Version 2.0 Part 1: Core Language*, March 2006. `http://www.w3.org/TR/2006/CR-wsdl20-20060327`.

[10] 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`.

[11] J. Colgrave and K. Januszewski. Using WSDL in a UDDI registry, version 2.0.2. Technical note, OASIS, 2004. `http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm`.

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

[13] David C. Fallside and Priscilla Walmsley. *XML Schema Part 0: Primer Second Edition*, October 2004. `http://www.w3.org/TR/xmlschema-0/`.

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

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

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

[17] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

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

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

APPENDIX

A. PROOFS

We give here the most important proof omitted in the main text. THEOREM 3.3

PROOF. We did not define $\overline{\sigma}$ formally here, so we will give an equivalent definition of $\ltimes$ not using it, which was also the definition used in the previous work, and just give the intuition of how we obtain it using $\overline{\sigma}$. The important property about $\overline{\sigma}$ is that its ready sets are defined as all the possible sets obtained by picking one name in each ready set of σ, and taking their co-names. This can be seen by looking at the definition of observable ready sets and thinking that we just exchange internal and external choices. Now if we look at definition 2.5 and apply it to $\sigma_c = \overline{\sigma_1}$ and $\sigma_s = \sigma_2$, the first condition says that any ready set of σ_2 contains at least a name from each ready set of $\overline{\sigma_1}$, which is equivalent to the fact that it contains a ready set of σ_1. Translation of condition 2 is straightforward, so we get that $\ltimes$ is the largest relation such that $\sigma_1 \ltimes \sigma_2$ implies :

1. $\forall \mathrm{R}_2, \sigma_2 \Downarrow \mathrm{R}_2 \Rightarrow \exists \mathrm{R}_1 \subseteq \mathrm{R}_2, \sigma_1 \Downarrow \mathrm{R}_1$ and
2. $\sigma_1 \stackrel{\alpha}{\longmapsto} \sigma_1'$ and $\sigma_2 \stackrel{\alpha}{\longmapsto} \sigma_2'$ implies $\sigma_1' \ltimes \sigma_2'$

Now let us prove that $\preceq$ is the transitive closure of the relation thus defined. Note that the condition on ready sets is the same in both relations, and that the condition on continuations for $\sigma_1 \preceq \sigma_2$ is a weakened version of the condition for $\sigma_1 \ltimes \sigma_2$, so obviously $\sigma_1 \ltimes \sigma_2$ implies $\sigma_1 \preceq \sigma_2$. So what we have to show is that two contracts related by $\preceq$ are also related by the transitive closure of $\ltimes$.

In fact we will show the following slightly stronger property :

$$\forall \sigma_1, \sigma_2 \in \Sigma, \sigma_1 \preceq \sigma_2 \Rightarrow \exists \sigma \in \Sigma, \sigma_1 \ltimes \sigma \text{ and } \sigma \ltimes \sigma_2$$

We do it by induction on the maximal depth $|\sigma_1|$ of σ_1. If this depth is zero, then $\sigma_1 = \mathbf{0}$ and we just take $\sigma = \mathbf{0}$ too.

Now suppose the property is true whenever $|\sigma_1| = n$, and let σ_1 be of depth $n+1$ and σ_2 any contract such that $\sigma_1 \preceq \sigma_2$. Let $(\mathrm{R}_i)_{i \in I}$ be the collection of ready sets of σ_2. By definition of $\preceq$, for every R_i we can find a ready set S_i of σ_1 such that: $\forall \alpha \in \mathrm{S}_i, \sigma_1(\alpha) \preceq \sigma_2(\alpha)$. Then, by induction hypothesis, for any α in any S_i we can find a contract σ_α such that $\sigma_1(\alpha) \ltimes \sigma_\alpha$ and $\sigma_\alpha \ltimes \sigma_2(\alpha)$, because $\sigma_1(\alpha)$ has

depth n. So we can pose the following[5]:

$$\sigma = \bigoplus_{i \in I} \sum_{\alpha \in \mathrm{S}_i} \alpha.\sigma_\alpha$$

Let us verify that $\sigma_1 \ltimes \sigma$: for condition 1, any ready set of σ is an S_i, which is a ready set of σ_1. Condition 2 is straightforward by definition of the σ_α. Now for $\sigma \ltimes \sigma_2$, we have that any ready set R_i of σ_2 contains S_i which is a ready set of σ, and condition 2 is again straightforward, so this closes the induction. □

THEOREM 3.12

PROOF. $\Rightarrow$. We do it by induction on the depth $|\sigma_1|$ of σ_1. If σ_1 has depth zero, then its only ready set is the empty set, so we have that any contract σ_c complying with σ_1 must have only empty ready sets, which means it complies with any σ_2.

Now suppose the implication is true for $|\sigma_1| = n$ and take σ_1 of depth $n+1$. Let $\sigma_2 \succeq \sigma_1$, let $\sigma_c \dashv \sigma_1$ and let A be the set of actions given by this compliance. We then pose the following:

$$\mathrm{B} = \{\alpha \in \mathrm{A} | \sigma_1 \stackrel{\alpha}{\longmapsto} \wedge \sigma_2 \stackrel{\alpha}{\longmapsto} \wedge \sigma_c \stackrel{\overline{\alpha}}{\longmapsto} \wedge \sigma_1(\alpha) \preceq \sigma_2(\alpha)\}$$

We want to use B to show compliance between σ_c and σ_2. Now let R_2 be a ready set of σ_2 and R_c a ready set of σ_c. Let R_1 be a ready set of σ_1 such that $\mathrm{R}_1 \subseteq \mathrm{R}_2$ and $\forall \alpha \in \mathrm{R}_1, \sigma_1(\alpha) \preceq \sigma_2(\alpha)$. We know that $\mathrm{R}_1 \cap \mathrm{A} \cap \mathsf{co}(\mathrm{R}_c) \neq \emptyset$. This last set is included in a ready set of each of our three contracts (R_1, R_2 and R_c), which means all three have a continuation for every name in this set, so in fact it is equal to $\mathrm{R}_1 \cap \mathrm{B} \cap \mathsf{co}(\mathrm{R}_c)$. Since this set is included in R_2, we also have $\mathrm{R}_2 \cap \mathrm{B} \cap \mathsf{co}(\mathrm{R}_c) \neq \emptyset$. Now for any name $\alpha \in \mathrm{B}$, we have that $\sigma_c(\overline{\alpha}) \dashv \sigma_1(\alpha)$, because $\mathrm{B} \subseteq \mathrm{A}$; and we also have that $\sigma_1(\alpha) \preceq \sigma_2(\alpha)$ by definition of B. Since $\sigma_1(\alpha)$ has depth n, we can conclude by induction hypothesis.

$\Leftarrow$. Let σ_1 and σ_2 be such that $\sigma_1 \not\preceq \sigma_2$. We show by induction on $|\sigma_2|$ that there exists a contract σ_c which complies with σ_1 but not with σ_2. If $|\sigma_2| = 0$, then σ_2 has an empty ready set, but σ_1 cannot have one. So we take one name in each ready set of σ_1 and define σ_c as the external choice of all their co-names followed by $\mathbf{0}$. Defining A as the only ready set of σ_c we get that $\sigma_c \dashv \sigma_1$, but obviously $\sigma_c \not\dashv \sigma_2$.

Now take σ_2 of depth $n+1$. Since $\sigma_1 \not\preceq \sigma_2$, there exists a ready set R_2 of σ_2 such that every ready set of σ_1 included in it contains a name α such that $\sigma_1(\alpha) \not\preceq \sigma_2(\alpha)$. We then construct σ in the following way: for each ready set of σ_1, we will define a contract. If this ready set is not included in R_2 we pick a name α in it which is not in R_2, and take the co-name of this name followed by any contract compliant with $\sigma_1(\alpha)$ (for example the null contract). If our ready set is included in R_2, we pick an α in it such that $\sigma_1(\alpha) \not\preceq \sigma_2(\alpha)$ and take the co-name of this name followed by a contract, obtained by induction hypothesis, which complies with $\sigma_1(\alpha)$ but not with $\sigma_2(\alpha)$. We then take σ_c as the external choice of all these contracts. Then with A defined as the only ready set of σ_c we get that $\sigma_c \dashv \sigma_1$. But there is no name in $\mathrm{R}_2 \cap \mathrm{A}$ such that $\sigma_c(\overline{\alpha}) \dashv \sigma_2(\alpha)$, so σ_c does not comply with σ_2. This closes the induction. □

[5]In fact, since we have not proved any commutativity and associativity properties for the external and internal choice operators, the formula does not define one single contract, but let us just say we take one of the possibilities.

THEOREM 3.17

PROOF. A maximal computation of the system $P \parallel Q$ is a sequence of systems $P_1 \parallel Q_1, \ldots, P_n \parallel Q_n$ such that $P_1 = P$, $Q_1 = Q$, for every $i = \{1, \ldots, n-1\}$ we have $P_i \parallel Q_i \longrightarrow P_{i+1} \parallel Q_{i+1}$, and $P_n \parallel Q_n \not\longrightarrow$. The proof is by induction on n.

If $n = 0$, then $P \parallel Q \not\longrightarrow$. We have two possibilities: if $P \stackrel{\alpha}{\not\longrightarrow}$ then by definition $P \dashv_S Q$. So let us suppose, by contradiction, that whenever $P \stackrel{\alpha}{\longrightarrow}$ we have $Q \stackrel{\overline{\alpha}}{\not\longrightarrow}$. Since $\vdash P : \sigma$ and $\vdash Q : \sigma'$ this means that for any ready set R of σ there is no ready set S of σ' such that $\overline{\mathrm{R}} \cap \mathrm{S} \neq \emptyset$, but this is absurd from the hypothesis that $\sigma \dashv_S \sigma'$.

If $n > 0$, assume that the theorem is true for any computation of length $n-1$. We have three cases:

($P \longrightarrow P'$) Assume $P' \vdash \sigma''$, then from the consistency condition 2 we have that $\sigma \Longrightarrow \sigma''$ and from Lemma 3.16 we obtain $\sigma'' \dashv_S \sigma'$. By the induction hypothesis we conclude that $P' \dashv_S Q$ hence $P \dashv_S Q$.

($Q \longrightarrow Q'$) Assume $Q' \vdash \sigma''$, then from the consistency condition 2 we have that $\sigma \Longrightarrow \sigma''$. Since $\sigma \Longrightarrow \sigma''$ implies $\sigma \sqsubseteq \sigma'$, from Theorem 3.5 we conclude that $P \dashv_S Q'$ hence $P \dashv_S Q$.

($P \stackrel{\alpha}{\longrightarrow} P'$ **and** $Q \stackrel{\overline{\alpha}}{\longrightarrow} Q'$) Assume that $P' \vdash \sigma''$ and $Q' \vdash \sigma'''$. From the consistency condition 3 we have that $\sigma \stackrel{\alpha}{\longmapsto\Longrightarrow} \sigma''$ and $\sigma' \stackrel{\overline{\alpha}}{\longmapsto} \sigma'''$. From Lemma 3.16 and Theorem 3.5 and from the definition of strong compliance we have that $\sigma'' \dashv_S \sigma'''$. The computation starting from $P' \parallel Q'$ has length $n-1$, by the induction hypothesis we have $P' \dashv_S Q'$ so we conclude $P \dashv_S Q$. □