A Theory of Contracts for Web Services

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

Contracts are behavioural descriptions of Web services. We devise a theory of contracts that formalises the compatibility of a client to a service, and the safe replacement of a service with another service. The use of contracts statically ensures the successful completion of every possible interaction between compatible clients and services.

The technical device that underlies the theory is the definition of *filters*, which are explicit coercions that prevent some possible behaviours of services and, in doing so, they make services compatible with different usage scenarios. We show that filters can be seen as proofs of a sound and complete subcontracting deduction system which simultaneously refines and extends Hennessy's classical axiomatisation of the must testing preorder. The relation is decidable and the decision algorithm is obtained via a cut-elimination process that proves the coherence of subcontracting as a logical system.

Despite the richness of the technical development, the resulting approach is based on simple ideas and basic intuitions. Remarkably, its application is mostly independent of the language used to program the services or the clients. We also outline the possible practical impact of such a work and the perspectives of future research it opens.

Keywords Web services, contracts, concurrency theory, CCS, must testing, type theory, subtyping, explicit coercions.

1. Introduction

The recent trend in Web Services is fostering a computing scenario where clients perform run time queries in search of 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) [14, 13, 12] 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) [2] extends WSDL contracts by allowing the description of arbitrary, possibly cyclic sequences of exchanged messages between communicating parties. Other languages, such as the Abstract Web Service Business Execution Language (WS-BPEL) [1], provide even more detailed descriptions by defining the subprocess structure, fault handlers, etc. While the latter descriptions are much too concrete to be used as contracts, they can be approximated and compared in terms of contracts that capture the external, observable behaviour of a service.

Documents describing contracts can be published in repositories (see [3, 15] for the case of WSDL and WSCL) so that Web services can be *searched* and *queried*. These two basic operations assume the existence of some notion of contract equivalence to perform service discovery in the same way as, say, type isomorphisms are used to perform library searches [27, 16]. The lack of a formal characterisation of contracts only permits excessively demanding notions of equivalence such as syntactical equality. In fact, it makes perfect sense to further relax the equivalence into a *subcontract preorder* (denoted by $\preceq$ in this paper), so that Web services exposing "larger" contracts can be *safely* returned as results of queries for Web services with "smaller" contracts.

In this work we develop a formal theory that precisely defines what "larger" and "smaller" mean, and which safety properties we wish to be preserved. Along the lines of [9] we describe contracts by a simple CCS-like syntax consisting of just three constructors: prefixing, 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, $\sigma \oplus \tau$ is the contract of services that may decide to implement either σ or τ, while $\sigma + \tau$ is the contract of services that according to their client's choice, will implement either σ or τ.

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. Actions can either represent *operations* or *message types*. As a matter of facts, contracts are behavioural types of processes that do not manifest internal moves and the parallel structure. They are *acceptance trees* in Hennessy's terminology [19, 20].

Contracts are then to be used to ensure that interactions between clients and services will always succeed. Intuitively, this happens if whenever a service offers some set of actions, the client either syn-

A very preliminary version of this work was presented on January 20, 2007, Nice, France, at PLAN-X 2007, the 5th ACM SIGPLAN Workshop on Programming Language Technologies for XML (no formal proceedings).

chronises with one of them (that is, it performs the corresponding co-action) or it terminates. The service 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 service.

Of course the client will probably be satisfied to interact with services that offer more than what the searched contract specifies. Intuitively we want to define an order relation on contracts $\sigma \preceq \tau$ such that every client complying with services implementing σ will also comply with services of contract τ. In particular, we would like the $\preceq$ preorder to enjoy some basic properties. The first one is that it should be safe to replace (the service exposing) a contract with a "more deterministic" one. 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 systematically offers $\overline{a}$. The second desirable property is that it should be safe to replace (the service exposing) a contract with another one that offers more capabilities. For instance, we expect $\overline{a} \preceq \overline{a} + \overline{b}.d$ 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 $\overline{b}.d$. If taken together, these 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 necessarily comply with $\overline{a} + \overline{b}.d$: if client and service synchronise on b, then the client will try to write on c while the service expects to read from d. Therefore, under this interpretation, $\preceq$ looks as not being transitive:

$$\overline{a} \oplus \overline{b}.c \preceq \overline{a} \quad \wedge \quad \overline{a} \preceq \overline{a} + \overline{b}.d \quad \Longrightarrow \quad \overline{a} \oplus \overline{b}.c \preceq \overline{a} + \overline{b}.d\,.$$

The problem can be solved by resorting to the theory of *explicit coercions* [5, 11, 28]. The flawed assumption of the approach described so far, which is the one proposed in [9], is that services are used carelessly "as they are". Note indeed that what we are doing here is to use a service of "type" $\overline{a} + \overline{b}.d$ where a service 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 services.[1] If we denote by $\mathrel{:>}$ the subtyping relation for services, then $\overline{a} \oplus \overline{b}.c \mathrel{:>} \overline{a} + \overline{b}.d$ and so what we implicitly did is to apply subsumption [8] and consider that a service 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, $\mathrel{:>}$) is not transitive. It rather resides in the use of subsumption, since this corresponds to the use of *implicit* coercions. Coercions have many distinct characterisations in the literature, but they all share the same underlying intuition that coercions are functions that embed objects of a smaller type into a larger type "without adding new computation" [11]. For instance it is well known that for record types one has $\{a{:}s\} \mathrel{:>} \{a{:}s; b{:}t\}$. 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.[2] 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 the function 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 computed at static time. Coming back to our example, the embedding of a service 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 service of the former type in order to use it in a context where a service of the latter type is expected. On the contrary, to embed a service of type $\overline{a} + \overline{b}.d$ into $\overline{a}$ we have to mask (at least) the $\overline{b}$ action of the service. So in order to use it in a context that expects a $\overline{a}$ service 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 service that implements $\overline{a} + \overline{b}.d$ by applying to this service 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.

Filters thus reconcile two requirements that were hitherto incompatible: On the one hand we wish to replace an old service by a new service that offers more choices (that is *width subtyping*, e.g. $\sigma \mathrel{:>} \sigma + \tau$) and/or longer interaction patterns (that is *depth subtyping*, e.g. $a \mathrel{:>} a.\sigma$) and/or is more deterministic (e.g. $\sigma \oplus \tau \mathrel{:>} \sigma$). On the other hand we want clients of the old service to seamlessly work with the new one.

Two observations to conclude this brief overview. First, the fact that we apply filters to services rather than to clients is just a presentational convenience: the same effect as applying to a service 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 interaction and a only at the second step of the interaction.

Outline of the presentation

We start by presenting the syntax of our contracts (§2.1), by showing how to use them to express WSDL and WSCL descriptions (§2.2), and by defining 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 de-

[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 [25]. This inversion corresponds to the duality between simulation and subtyping, viz. between observers and observed behaviours.

[2] In the case of typed lambda calculus coercions are formally characterised by the fact that their type erasure is η-equivalent to the identity function, but in general coercions may not be the identity function [11].

scribe the relation with the must preorder (§3.1). This subcontract relation induces a notion of weak compliance which suggests that non-interference of unexpected actions can be ensured by coercion functions, which we dub *filters*. By shielding the actions at issue, a filter embeds a service into the "world" of its expected clients. We prove that our subcontract relation can be expressed in terms of filters and of the must preorder and we provide a sound and complete deduction system for the subcontract relation where filters play the role of "proofs" (§3.2). The subcontract relation is shown to be decidable via the definition of a sound and complete algorithmic deduction system (§3.3). Finally, we relate our contract language with a generic process language (of which we only require the existence of a labelled transition system and of a type system associating contracts with processes). The soundness of our theory of contracts is proved by showing 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 (§3.4). A conclusion recaps our work and hints at possible tracks of future research (§4).

Note for the reviewers: Proofs of lemmas and theorems have been omitted because of space limits. They can be found in the full version of the paper, which is available at `http://www.sti.uniurb.it/padovani/Papers/filtered_contracts.pdf`.

Related work

This research was initially inspired by "CCS without $\boldsymbol{\tau}$'s" [26] and by Hennessy's model of acceptance trees [19, 20]. Our contracts are an alternative representation of acceptance trees. The works that are more closely related to ours are by Carpineti *et al.* [9] and the ones on *session types*, especially [18] by Gay and Hole. In [9] the subcontract relation exhibits all of the desirable properties illustrated in the introduction, but subcontracting essentially stopped at the problem of transitivity. In that work compliance was a syntactic notion and contracts lacked a semantic characterisation.

Session types were introduced by Honda *et al.* [21, 29, 22] in the context of the π-calculus. These are used to type special channels through which several different messages may be exchanged in sequence according to a given protocol. Such a session channel can be seen as a client-service connection, and the session type is the analogous of our contract as it describes which actions the processes may perform through this channel. However, session types have the important restriction, if compared with contracts, that only one part has the floor at a given time: whenever a process performs an internal choice it has to indicate explicitly which path of interaction it has chosen, and the other process has to be waiting for this indication. Thus there is no way of mixing internal and external choices, and two processes like $a + b$ and $\overline{a} + \overline{b}$ do not interact successfully (because nobody has the floor, so no communication can happen). Subtyping for the session types has been studied by [18], but because of the aforementioned restriction, the transitivity problem we address in this paper does not exist for them: internal and external choices can never be related (hence, $a \oplus b \preceq a + b$ does *not* hold). Carbone *et al.* [6, 7] define a global calculus to describe choreographies of Web services, and an end-point projection to infer descriptions of individual processes from the global description, both of which are heavily based on session types. Our approach is different since our typical application for Web services is searching for a service compatible with a given protocol *from the client point of view*: in particular, we want depth subtyping (a service doing further actions after the client has successfully terminated is compatible with this client), which does not hold for session types. We believe that our theory is more basic than the theory of session types and that it can be fruitfully used to enrich the latter.

Finally, there is an important connection with the work of Laneve and Padovani [23], who propose an approach where contracts are "statically" filtered. Filtering is materialised by associating each contract with a *static interface* that declares the only visible actions of the contract and that does not change over time. Our approach using explicit coercions subsumes static interfaces and allows us to further relax the subcontract relation. The work [23] and older ones on the testing framework have shown that the subcontract relation is not affected in its essence by recursion, and the desirable properties we mentioned in the introduction and formalised later in the deduction systems still hold. For the sake of brevity, in this paper we only consider finite contracts without recursion, but the extension of our contracts and filters to the non-finite case, although it involves significant technicalities, is conceptually straightforward. The other aspects investigated in [23] regard contract duality and the application of contracts to choreographies of Web services, while we focus on the theory of dynamically filtered contracts, on its algorithmic aspects, and on language neutrality.

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.

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

where $\mathbf{0}$ is the contract of services that do not perform any action (the other constructions were already explained in the introduction). We follow the standard convention of omitting trailing $\mathbf{0}$'s.

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) [14, 4, 12, 13] permits to describe and publish abstract and concrete descriptions of Web services. Such descriptions include the schema [17] 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 interaction patterns (called *message exchange patterns* or MEPs in version 2.0 of WSDL) determining the order and direction of the exchanged messages. In particular, WSDL 2.0 predefines four message exchange patterns 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. When the MEP is `inOnly` or `robustInOnly`, communication is basically *asynchronous*: the client can only send an `In` message containing the request. If the pattern is `robustInOnly` the service may optionally send back a `Fault` message indicating that an error has occurred. When the MEP is `inOut` or `inOptOut`, communication is basically *synchronous*: the client sends an `In` message containing the request and the service sends back either an `Out` message containing the response or a `Fault` message. If the pattern is `inOptOut`, then the `Out` message is optional. These

four patterns can be encoded in our contract language as follows:

$$\begin{aligned} \texttt{inOnly} &= \texttt{In} \\ \texttt{robustInOnly} &= \texttt{In}.(\mathbf{0} \oplus \overline{\texttt{Fault}}) \\ \texttt{inOut} &= \texttt{In}.(\overline{\texttt{Out}} \oplus \overline{\texttt{Fault}}) \\ \texttt{inOptOut} &= \texttt{In}.(\mathbf{0} \oplus \overline{\texttt{Out}} \oplus \overline{\texttt{Fault}}) \end{aligned}$$

It is worth noticing that, intuitively, a client that is capable of invoking a service whose MEP is `robustInOnly` will also interact successfully with a service whose MEP is `inOnly`. Indeed, such client must be able to handle both a communication that terminates *and* a `Fault` message. Similary, a client that is capable of invoking a service whose MEP `inOptOut` will also interact successfully with services whose MEP is either `inOut`, or `robustInOnly`, or even `inOnly`. On the other hand, a client that interacts with a service whose MEP is `inOut` will not (always) interact successfully with a service whose MEP is `inOptOut`. The client assumes that it will always receive either an `Out` or a `Fault` message, but `inOptOut` does not give this guarantee.

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 [2] 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:

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

Notice that unlabeled transitions in Figure 1 correspond to external choices in σ, whereas labeled transitions correspond to internal choices.

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:

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

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 $\sigma \stackrel{\alpha}{\not\longmapsto}$ be the least relation such that:*

$$\begin{aligned} \mathbf{0} &\stackrel{\alpha}{\not\longmapsto} \\ \beta.\sigma &\stackrel{\alpha}{\not\longmapsto} && \text{if } \alpha \neq \beta \\ \sigma \oplus \tau &\stackrel{\alpha}{\not\longmapsto} && \text{if } \sigma \stackrel{\alpha}{\not\longmapsto} \text{ and } \tau \stackrel{\alpha}{\not\longmapsto} \\ \sigma + \tau &\stackrel{\alpha}{\not\longmapsto} && \text{if } \sigma \stackrel{\alpha}{\not\longmapsto} \text{ and } \tau \stackrel{\alpha}{\not\longmapsto} \end{aligned}$$

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

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

$$\frac{\sigma \stackrel{\alpha}{\longmapsto} \sigma' \quad \tau \stackrel{\alpha}{\longmapsto} \tau'}{\sigma + \tau \stackrel{\alpha}{\longmapsto} \sigma' \oplus \tau'} \qquad \frac{\sigma \stackrel{\alpha}{\longmapsto} \sigma' \quad \tau \stackrel{\alpha}{\not\longmapsto}}{\sigma + \tau \stackrel{\alpha}{\longmapsto} \sigma'}$$

$$\frac{\sigma \stackrel{\alpha}{\longmapsto} \sigma' \quad \tau \stackrel{\alpha}{\longmapsto} \tau'}{\sigma \oplus \tau \stackrel{\alpha}{\longmapsto} \sigma' \oplus \tau'} \qquad \frac{\sigma \stackrel{\alpha}{\longmapsto} \sigma' \quad \tau \stackrel{\alpha}{\not\longmapsto}}{\sigma \oplus \tau \stackrel{\alpha}{\longmapsto} \sigma'}$$

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 [24]. 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 branch 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 $\sigma \stackrel{\alpha}{\longmapsto} \sigma'$.*

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 $\mathscr{P}_f(\mathscr{N} \cup \overline{\mathscr{N}})$ be the set of finite parts of $\mathscr{N} \cup \overline{\mathscr{N}}$, called* ready sets. *Let also $\sigma \Downarrow$ R*

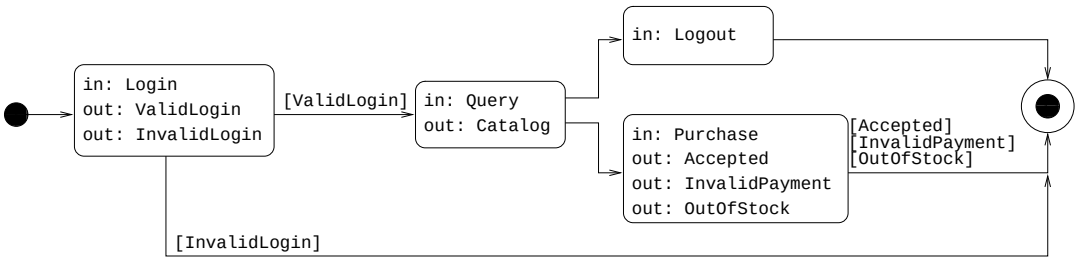

Figure 1. Contract of a simple e-commerce service as a WSCL diagram.

be the least relation between contracts σ in Σ and ready sets R *in $\mathscr{P}_f(\mathscr{N} \cup \overline{\mathscr{N}})$ such that:*

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

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

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 §2.2. This first requires the precise definition of *compliance*. Recall that, intuitively, the behaviour of a client complies with the behaviour of a service if for every set of actions that the service may offer, the client either synchronises with one of them, or it terminates successfully. The behaviour of clients, as well as the one of services, is described by contracts. Therefore we need to define when a contract ρ describing the behaviour of a client complies with a contract σ describing the behaviour of a service. For this we reserve a special action $\mathtt{e}$ (for "end") that can occur in client contracts and that represents the ability of the client to successfully terminate. Then we require that, whenever no further interaction is possible between the client and the service, the client be in a state where this action is available.

DEFINITION 2.5 (STRONG COMPLIANCE). *$\mathscr{C}$ is a* strong compliance relation *if $(\rho, \sigma) \in \mathscr{C}$ implies that:*

1. *$\rho \Downarrow \mathrm{R}$ and $\sigma \Downarrow \mathrm{S}$ implies either $\mathtt{e} \in \mathrm{R}$ or $\mathtt{co}(\mathrm{R}) \cap \mathrm{S} \neq \emptyset$, and*
2. *$\rho \stackrel{\overline{\alpha}}{\longmapsto} \rho'$ and $\sigma \stackrel{\alpha}{\longmapsto} \sigma'$ implies $(\rho', \sigma') \in \mathscr{C}$.*

We use $\dashv$ to denote the largest strong compliance relation.

In words the definition above states that a client of contract ρ is compliant with a service of contract σ if (1) for every possible combination S and R of the independent choices of the service and the client, either there is an action in the client choice that can synchronise with an action among those offered by the service ($\mathtt{co}(\mathrm{R}) \cap \mathrm{S} \neq \emptyset$) or the client terminates successfully ($\mathtt{e} \in \mathrm{R}$), and (2) whenever a synchronisation happens, the continuation of the client after it is compliant with the continuation of the service ($(\rho', \sigma') \in \mathscr{C}$).

Once we have such a definition it is natural to define the subcontract relation in terms of compliance. Intuitively, (client) contracts are seen as "tests" for comparing (service) contracts. Two (service) contracts are related if so are the sets of (client) contracts compliant with them [25].

DEFINITION 2.6 (STRONG SUBCONTRACT). *The contract σ is a* strong subcontract *of the contract τ, written $\sigma \sqsubseteq \tau$, if and only if for all ρ we have $\rho \dashv \sigma$ implies $\rho \dashv \tau$. We write $\sigma \simeq \tau$ if $\sigma \sqsubseteq \tau$ and $\tau \sqsubseteq \sigma$.*

This definition corresponds to giving a set theoretic semantics to service contracts which are thus interpreted as the set of their compliant clients. Thus $\sqsubseteq$ is interpreted as set-theoretic inclusion.

As usual with testing semantics, it is hard to establish a relationship between two contracts because the set of clients that are strongly compliant is infinite. A direct definition of the preorder is therefore preferred:

DEFINITION 2.7. *$\mathscr{S}$ is a coinductive strong subcontract relation if $(\sigma, \tau) \in \mathscr{S}$ implies that*

1. *$\tau \Downarrow \mathrm{R}$ implies that there exists $\mathrm{S} \subseteq \mathrm{R}$ such that $\sigma \Downarrow \mathrm{S}$, and*
2. *$\tau \stackrel{\alpha}{\longmapsto} \tau'$ implies $\sigma \stackrel{\alpha}{\longmapsto} \sigma'$ and $(\sigma', \tau') \in \mathscr{S}$.*

THEOREM 2.8. *$\sqsubseteq$ is the largest coinductive strong subcontract relation.*

It turns out that the relation $\sqsubseteq$ is the *must testing preorder* as defined in [25] (a proof can be found in [23], where a different albeit equivalent notion of strong compliance is used). This relation is well studied and it enjoys interesting properties, in particular it is a precongruence with respect to prefixing, internal and external choices, and also $a \oplus b \sqsubseteq a$, which is one of the desirable properties for $\preceq$, holds. However $\sqsubseteq$ is stronger than $\preceq$ since, for example, $\overline{a} \not\sqsubseteq \overline{a} + \overline{b}$. Indeed $a.\mathtt{e} + b \dashv \overline{a}$ but $a.\mathtt{e} + b \not\dashv \overline{a} + \overline{b}$. In general, the must preorder allows neither width nor depth extensions of contracts.

In previous work [9] an attempt was made to directly relate two contracts σ and τ depending on their form, rather than on the sets of their clients. Let $\mathtt{dual}(\sigma)$ denote the dual contract of σ which, roughly, is obtained by replacing in σ every action by its coaction, $\mathbf{0}$ by $\mathtt{e}$, every internal choice by an external one, and viceversa (the formal definition is slightly more involved and requires first to transform σ into the normal form of Definition 3.10 and then apply the transformation described above; see [9] for details). Intuitively $\mathtt{dual}(\sigma)$ denotes the contract of a "canonical" client complying with σ services. Then one can define a new relation on service contracts as:

$$\sigma \ltimes \tau \stackrel{\text{def}}{\iff} \mathtt{dual}(\sigma) \dashv \tau \qquad (1)$$

In words, a contract σ is a subcontract of τ if and only if its canonical client complies with τ.

This relation is *nearly* what we are looking for. For instance now we have $a \oplus b.c \ltimes a$ and $a \ltimes a + b.d$, since $\mathtt{dual}(a \oplus b.c) = \overline{a}.\mathtt{e} + \overline{b}.\overline{c}.\mathtt{e} \dashv a$ and $\mathtt{dual}(a) = \overline{a}.\mathtt{e} \dashv a + b.d$.

Unfortunately, $\ltimes$ is not a preorder since transitivity does not hold: $\overline{a}.\mathtt{e} + \overline{b}.\overline{c}.\mathtt{e} \not\asymp a + b.d$ implies that $a \oplus b.c \not\ltimes a + b.d$. The reason for such a failure is essentially due to the fact that in establishing $a \oplus b.c \ltimes a$ and $a \ltimes a + b.d$ we are restricting compliance to conversations in which no synchronisation on the name b happens. While contracts account for non-determinism that is internal to each process—being it a client or a service—, they cannot handle the "system" non-determinism that springs from process synchronisation. In the example above, the failure results from the interaction of two external choices, $\overline{a}.\mathtt{e}+\overline{b}.\overline{c}.\mathtt{e}$ and $a+b.d$, which yields non-determinism at system level and which does not prevent *a priori* a synchronisation on the b name. By preventing the synchronisation on the name b, the client $a.\mathtt{e} + b.\overline{c}.\mathtt{e}$ can terminate successfully.

In summary, the strong subcontract relation implements a safe substitutability relation for services that *are* compatible, but is excessively demanding because it takes into account every possible synchronisation. Our theory of contracts will define a safe substitutability relation for services that *can be made* compatible.

3. A theory of contracts

At the end of the previous section we said that we wanted a subcontract relation $\sigma \preceq \tau$ such that a service with contract τ *can be made* compatible with a service with contract σ. 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. In fact, we cannot hope to affect in any way the internal non-determinism of a service as the service is typically considered as an unmodifiable black box. Instead we look for transformations that embed a τ service in a world of σ clients so that such clients will perceive their interaction as being carried over a service with contract σ (or possibly a more deterministic one). Roughly speaking we want to filter out all behaviours of the τ contract that do not belong to the possible behaviours of σ world, and leave the others unchanged. This is, precisely, the characterisation of an explicit coercion from τ to σ (recall that the subcontract relation is the inverse of a service subtyping relation; *c.f.* Footnote 1): an embedding function that maps possible behaviours of τ into the same behaviours of σ (thus, it does not add new computation).

3.1 Weak subcontract relation

The idea is that $\sigma \preceq \tau$ if there exists some (possibly empty) set of actions belonging to the world of τ that, if shielded, can make a τ service appear as a σ service. This is formalised by the following definition:

DEFINITION 3.1 (WEAK SUBCONTRACT). *$\mathscr{W}$ is a* weak subcontract relation *if $(\sigma,\tau) \in \mathscr{W}$ implies that (1) if $\tau \Downarrow \mathrm{R}$, then there exists $\mathrm{S_R} \subseteq \mathrm{R}$ such that $\sigma \Downarrow \mathrm{S_R}$ and (2) for all $\alpha \in \mathrm{S_R}$ we have $(\sigma(\alpha), \tau(\alpha)) \in \mathscr{W}$.*

We denote by $\preceq$ the largest weak subcontract relation.

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

First of all notice that the weak subcontract relation includes the strong one (condition (1) is the same and condition (2) is weaker), so that, for example, $a \oplus b.c \preceq a$ holds. Additionally, we also have $a \preceq a + b.d$ since a service with contract $a + b.d$ can be made to behave as a service with contract a by filtering out the b action. On the other hand, $a \not\preceq a \oplus b.c$ since there is no way to make $a \oplus b.c$ behave as a by simply filtering out actions (filtering out the b action from $a \oplus b.c$ yields $a \oplus \mathbf{0}$, not a). Finally, we also have $a \oplus b.c \preceq a + b.d$, again by filtering out the b action. In this case, the filtered service ($a + b.d$) is not made equivalent to the smaller service ($a \oplus b.c$) but rather to one of its more deterministic behaviours (a).

3.1.1 Weak compliance

In contrast with the "strong" case, for the weak subcontract relation it was more intuitive to provide its coinductive characterisation first. We now face the problem of understanding which notion of compliance induces the weak subcontract relation. As we will see, this is an essential intermediate step as it provides the necessary insight for devising the practical solution to the problems described in §2.4.

DEFINITION 3.2 (WEAK COMPLIANCE). *$\mathscr{D}$ is a* weak compliance relation *if $(\rho,\sigma) \in \mathscr{D}$ implies that there exists a finite set of actions $\mathrm{A} \subseteq \mathscr{N} \cup \overline{\mathscr{N}}$ such that:*

1. *$\rho \Downarrow \mathrm{R}$ and $\sigma \Downarrow \mathrm{S}$ implies $\mathtt{e} \in \mathrm{R}$ or $\mathtt{co}(\mathrm{R}) \cap \mathrm{A} \cap \mathrm{S} \neq \emptyset$, and*
2. *$\alpha \in \mathrm{A}$, $\rho \overset{\overline{\alpha}}{\longmapsto} \rho'$ and $\sigma \overset{\alpha}{\longmapsto} \sigma'$ implies $(\rho', \sigma') \in \mathscr{D}$.*

We denote by $\dashv\!\!\mid$ the largest weak compliance relation.

Note how the existence of the set A in the above definition must be *independent* of the ready sets of the client and of the service. This reflects the fact that the internal non-determinism of the interacting parties cannot be affected.

The following theorem proves that $\dashv\!\!\mid$ is the compliance relation inducing $\preceq$.

THEOREM 3.3. *$\sigma \preceq \tau$ if and only if for all ρ, $\rho \dashv\!\!\mid \sigma$ implies $\rho \dashv\!\!\mid \tau$.*

3.1.2 Comparison with other relations

In §2.4 we said that the relation $\ltimes$ defined by equation (1) 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 3.4. *The subcontract relation $\preceq$ is the transitive closure of $\ltimes$.*

For what concerns the inclusion of the strong relation in the weak one note that if we compare Definition 3.1 with Definition 2.7, we see that they differ on the set of α's 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 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.

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

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

(MUST) $I_\sigma \vee I_\tau : \sigma \oplus \tau \leq \sigma$

(DEPTHEXT) $\mathbf{0} : \mathbf{0} \leq \sigma$

(WEAKENING) $$\frac{f : \sigma \leq \tau \quad g \wedge I_\tau \leqslant f}{f \vee g : \sigma \leq \tau}$$

(TRANSITIVITY) $$\frac{f : \sigma \leq \sigma' \quad g : \sigma' \leq \sigma''}{f \wedge g : \sigma \leq \sigma''}$$

(PREFIX) $$\frac{f : \sigma \leq \tau}{\alpha.f : \alpha.\sigma \leq \alpha.\tau}$$

(INTCHOICE) $$\frac{f : \sigma \leq \sigma' \quad f : \tau \leq \tau'}{f : \sigma \oplus \tau \leq \sigma' \oplus \tau'}$$

(EXTCHOICE) $$\frac{f : \sigma \leq \sigma' \quad f : \tau \leq \tau'}{f : \sigma + \tau \leq \sigma' + \tau'}$$

Table 1. Deduction system for the weak subcontract relation.

3.2 Filters

A filter is the specification of a set of actions that are allowed at a certain time, along with the continuation filters that are applied after an 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:

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

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

$$\begin{array}{rcll} f(\mathbf{0}) & = & \mathbf{0} & \\ f(\alpha.\sigma) & = & \mathbf{0} & \text{if } f \stackrel{\alpha}{\not\longmapsto} \\ f(\alpha.\sigma) & = & \alpha.f_\alpha(\sigma) & \text{if } f \stackrel{\alpha}{\longmapsto} f_\alpha \\ f(\sigma + \tau) & = & f(\sigma) + f(\tau) & \\ f(\sigma \oplus \tau) & = & f(\sigma) \oplus f(\tau) & \end{array}$$

Filter application is monotone with respect to the strong subcontract preorder. This property, which is fundamental in proving most of the results that follow, guarantees that equivalent contracts remain equivalent if filtered in the same way.

PROPOSITION 3.5. *$\sigma \sqsubseteq \tau$ implies $f(\sigma) \sqsubseteq f(\tau)$.*

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

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

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 $f = \overline{a}$, 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.

3.2.1 Deduction system for $\preceq$

Filters can also be used as proofs (in the sense of the Curry-Howard isomorphism) for the weak subcontract relation. More specifically, the idea is to devise a deduction system within which a derivable judgement of the form $f : \sigma \leq \tau$ implies that $\sigma \preceq \tau$, and f is a filter that embeds the service with contract τ into the world of σ-compliant clients.

The definition of such deduction system requires a few auxiliary notions. First we have to define the "identity" filter, that is the one that proves isomorphic (with respect to an interpretation of filters as morphisms) contracts.

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

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

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

Next, we define two basic operations for combining filters. Intuitively, given a derivation tree for the judgement $f : \sigma \leq \tau$, such operations allow us to show how the filter f is built step-by-step, according to the structure of the derivation.

DEFINITION 3.8. *Let f and g denote the filters $\coprod_{\alpha \in \mathrm{A}} \alpha.f_\alpha$ and $\coprod_{\alpha \in \mathrm{B}} \alpha.g_\alpha$ respectively. Then the* conjunction *and* disjunction *of f and g are respectively defined as follows:*

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

Finally, we need a way for comparing filters. Filters can be compared according to the actions that they let pass. In the deduction system the need for comparing filters arises naturally in the weakening rule, where we want to replace a filter with a "larger" one (a

filter that allows more actions). This can be done safely only if the larger filter does not thwart the functionality of the original filter by re-introducing actions that must be kept hidden. The filter pre-order will also be fundamental in §3.3, in order to define the "best" filter that proves $\sigma \preceq \tau$.

DEFINITION 3.9. *The ordering relation on filters $f \leqslant g$ is the least relation such that $\coprod_{\alpha\in A} \alpha.f_\alpha \leqslant \coprod_{\beta\in B} \beta.g_\beta$ implies $A \subseteq B$ and for every $\alpha \in A$, $f_\alpha \leqslant g_\alpha$.*

Filters can be seen as n-ary trees with edges labelled by actions, each node having at most one outgoing edge labelled by a given action. The ordering we just introduced is nothing but tree inclusion where we consider that all trees share the same root. It is useful to notice that the syntactical "conjunction" and "disjunction" in Definition 3.8 can be alternatively defined in a natural way using the ordering: the conjunction of two filters is the largest part common to both trees, that is, their greatest lower bound:

$$f_1 \geqslant g \text{ and } f_2 \geqslant g \iff (f_1 \wedge f_2) \geqslant g \tag{2}$$

Similarly, the disjunction of two filters is the tree obtained by merging the two initial trees, that is their least upper bound:

$$f_1 \leqslant g \text{ and } f_2 \leqslant g \iff (f_1 \vee f_2) \leqslant g \tag{3}$$

A further interpretation of filters is as prefix-closed regular languages of strings of actions. Then, filter conjunction and disjunction correspond to language intersection and union, respectively, whereas the filter ordering is set inclusion (notice that the intersection and the union of prefix-closed sets is again prefix-closed).

Table 1 defines the deduction system for $\preceq$. In the table we use a single axiom $\sigma = \tau$ as a shorthand for two axioms $I_\sigma : \sigma \leq \tau$ and $I_\tau : \tau \leq \sigma$. The equalities and rule (MUST) are well known since they fully characterise the strong compliance relation, which coincides with the must preorder (see [25, 20]). Notice that in the rule (MUST) no action needs to be filtered out. In fact, this is the only axiom for safely enlarging a contract without the intervention of any filter (which is expected since this axiom characterises strong compliance, where filters are not needed). Rule (DEPTHEXT) formalises *depth* extension of contracts, where a contract can be prolonged if no action is made visible from the continuation. Rule (WEAKENING) shows how to safely enlarge a filter f to $f \vee g$: the premise $g \wedge I_\tau \leqslant f$ states that g may allow actions not allowed by f, provided that such actions are not those that have been hidden for the purposes of proving $f : \sigma \leq \tau$. Rule (TRANSITIVITY) is standard and the resulting filter is the composition filter. Three forms of (limited) pre-congruence follow. Rule (PREFIX) is standard and poses no constraints. Rules (INTCHOICE) and (EXTCHOICE) state the limited precongruence property for internal and external choices, respectively. The fundamental constraint is that two contracts combined by means of $\oplus$ or $+$ can be enlarged, provided that they can be filtered in the same way. This requirement has an intuitive explanation: the filter that mediates the interaction of a client with a service is unaware of the internal choices that have been taken by the parties at a branching point. So, it must be possible to use *the same* filter that works equally well in all branches in order for the branches to be enlarged.

By combining the rules (DEPTHEXT), (WEAKENING), and (EXTCHOICE) it is easy to derive a further rule, which formalises *width* extension of contracts:

$$\text{(WIDTHEXT)}\quad \frac{I_\sigma \wedge I_\tau \leqslant \mathbf{0}}{I_\sigma : \sigma \leq \sigma + \tau}$$

Basically (WIDTHEXT) states that a service can be extended so that it provides more capabilities, provided that such capabilities are disjoint from those that were available before the extension.

3.2.2 Properties

The deduction system we devised in the previous section is sound and complete with respect to $\preceq$ *and* the set of filters, in the sense that it proves all and only the pair of contracts that are related according to Definition 3.1, and for any such pair it deduces all and only the filters that validate the pair according to Theorem 3.6.

While the soundness of the deduction system can be easily established, its completeness is less immediate, but the proof of this fact follows a standard pattern: completeness is proved for a restricted class of contracts which are said to be in some normal form and then it is shown that it is always possible to transform an arbitrary contract to an equivalent one which is in normal form by using the axioms. Although in this version of the paper the proofs of theorems are omitted, we nevertheless introduce here the normal form of contracts. The same normal form will be necessary anyway in §3.3 for defining the algorithmic version of the deduction system.

As regards the actual definition of the normal form, we can notice that it is always possible to add new ready sets to a given contract σ without altering its semantics (according to $\simeq$), so long as I_σ does not change and the new ready sets contain older ones: for example, $\sigma \oplus \tau \simeq \sigma \oplus \tau \oplus (\sigma + \tau)$. Now we can see that, if we saturate the set of ready sets of a contract by adding to it every possible ready set meeting the conditions above, we can build a unique (up to commutativity and associativity) normal form for each equivalence class. This normal form is defined as follows:

DEFINITION 3.10 (NORMAL FORM [20]). *For any contract σ, we define its saturated set of ready sets:*

$$\mathscr{R}(\sigma) \stackrel{\text{def}}{=} \{R \subseteq \textstyle\bigcup_{\sigma\Downarrow S} S \mid \exists S, \sigma \Downarrow S \wedge S \subseteq R\}$$

The normal form of σ is then defined up to associativity and commutativity of the choices by the following recursive expression:

$$\mathtt{nf}(\sigma) \stackrel{\text{def}}{=} \bigoplus_{R\in\mathscr{R}(\sigma)} \sum_{\alpha\in R} \alpha.\mathtt{nf}(\sigma(\alpha))$$

the empty external choice being defined as $\mathbf{0}$ (it is not necessary to define the empty internal choice, because any contract has at least one ready set).

Normal forms can be used as the canonical representations of classes of the equivalence relation $\simeq$:

PROPOSITION 3.11. $\sigma \simeq \mathtt{nf}(\sigma)$.

The normal form enjoys also the following important properties: (1) In a given mix of internal and external choices (either at top-level or under a given sequence of prefixes), a prefix α is always followed by the exact same continuation. (2) If σ and τ are two normal form contracts such that $\sigma \sqsubseteq \tau$, condition (1) of the strong subcontract relation holds if and only if every ready set of τ is also a ready set of σ. These two properties lead to the fact that two equivalent normal forms are syntactically equal up to commutativity and associativity of the choice operators.

We now possess all the technical tools to prove that the deduction system shown in Table 1 is sound and complete for $\preceq$ and the sets of filters that prove it.

THEOREM 3.12. *$f : \sigma \leq \tau$ if and only if $\sigma \sqsubseteq f(\tau)$.*

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

COROLLARY 3.13.

$$\rho \dashv\vdash \sigma \iff \exists \tau \preceq \sigma, \rho \dashv \tau \quad (4)$$
$$\iff \exists f, \rho \dashv f(\sigma) \quad (5)$$

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 service and a weakly compliant client, every interaction between them mediated by the filter that proves the weak compliance (Theorem 3.13(2)) will be successful (the client terminates). This will be formally stated in §3.4.

3.3 Algorithmic deduction system

We introduced a device, filters, that allows us to transform a weak subcontract or compliance relation into a strong one by shielding the incompatible actions. The next step is to infer filters algorithmically, so that the weak relations can be used in practice.

As usual the process of finding a decision algorithm for a containment relation corresponds to a cut-elimination process (the cut here being the (TRANSITIVITY) rule in Table 1), which amounts to finding a canonical proof for each provable relation. In other terms, we have to associate every provable weak subcontracting relation with a canonical filter that represents all other possible proofs. In order to choose a canonical filter, we have to solve two potential problems. First, there usually are several filters that work with a given relation. For example, to show that $a \oplus b \preceq a + b$, we can either let pass only a, only b, or both. The best solution here is to let pass both, because we do not want to shield out actions that cannot cause any harm. This example suggests the definition of a notion of "better filter", that is, of a partial order on filters that determines which filter is better to use, and such partial order is exactly $\leqslant$ (Definition 3.9). The second problem is that in the example above a filter that lets a, b, and, say, c pass will work as well. The intuition here is that the filter that lets *just* a and b pass is better since allowing any action besides a and b to pass is useless. This suggests the definitions of a notion of "filter relevance", to single out filters that do not contain useless actions.

The subcontracting algorithm will pick up, among all the possible filter for a given relation, the "best relevant" filter that proves it.

3.3.1 Filter relevance

In order to determine the property of "relevance" we have to better understand the role played by the identity filters. It may be noted that the identity filter of a given contract is exactly the tree of all possible sequences of actions that the contract can do before reducing to $\mathbf{0}$, without distinguishing between internal and external choices. This is embodied by the $\vee$ operator on filters which is a unique choice operator representing both kinds of choice, as the following relation shows:

$$I_{\sigma \oplus \tau} = I_{\sigma + \tau} = I_\sigma \vee I_\tau \quad (6)$$

Note that if σ and τ share common actions in their outermost prefixes, the continuations of both filters after this action are correctly merged by the disjunction operator.

The tree of an identity filter accurately represents the idea we mentioned in the introduction of a contract's "world": the sets of actions the contract knows of at each step of an interaction. A filter $f : \sigma \leq \tau$ embeds τ services into the "world" of σ: then the intuition is that to be relevant f must be defined (only) on the "world" of τ, world that is represented by I_τ. Indeed, applying to τ the filter f or the filter $f \wedge I_\tau$ give the same result, thus the part of f that is not in $f \wedge I_\tau$ is irrelevant (and this is why there is no greatest filter corresponding to a given relation in the absolute). Thus we will say that a filter f is *relevant* with respect to a relation $\sigma \preceq \tau$ if it is smaller than I_τ.

Now if we restrict ourselves to relevant filters we can have another interesting upper bound: if we look at condition (2) of the strong subcontract relation, we see that, at each step, every action available to the greater contract has to be available also to the smaller one. This exactly means that the greater contract has a smaller tree, and thus we have (by noticing that $I_{f(\sigma)} = f \wedge I_\sigma$):

$$\text{if } \sigma \sqsubseteq f(\tau) \text{ and } f \leqslant I_\tau \text{ then } f \leqslant I_\sigma \quad (7)$$

Thus relevant filters that prove a relation have to be smaller than the identity filters of *both* contracts.

We now would like to find the greatest relevant filter that proves a given relation. Note that projecting on $I_\sigma \wedge I_\tau$ itself is not necessarily enough to make the relation work, because of ready sets: it might be necessary to project on something smaller to prevent a wrong branch to be taken, for example in $a \oplus b.(a+b) \preceq a + b.(a \oplus b)$, the initial b has to be filtered out even if the trees are the same, because its continuation in the right contract has incompatible ready sets. However, the following important relation holds:

$$\text{if } \sigma \sqsubseteq f(\tau) \text{ and } \sigma \sqsubseteq g(\tau) \text{ then } \sigma \sqsubseteq (f \vee g)(\tau) \quad (8)$$

meaning that if we can make the relation work either by selecting some branches or by selecting some other branches, then it will still work if we take all these branches at once. This shows that, if $\sigma \preceq \tau$ holds, there will be a greatest *subtree* of τ that makes the relation work: even if there is no greatest filter in the absolute, we can take the disjunction of all filters less than I_τ that work (there are a finitely many). This filter, which is the least upper bound of all relevant filters that prove $\sigma \preceq \tau$, is the one we choose as canonical.

3.3.2 Algorithm

The last step is to define an algorithm for building the canonical filter of a relation. The monotonicity of filters (Proposition 3.5) and the soundness and completeness of the deduction system (Theorem 3.12) ensure that filters prove subcontracting modulo equivalence, that is if $f : \sigma \leq \tau$, then $f : \sigma' \leq \tau'$, for any $\sigma' \simeq \sigma$, $\tau' \simeq \tau$. Since a contract is equivalent to its normal form (Proposition 3.11), then the set of filters that prove $\sigma \preceq \tau$ is the same as the set of those that prove $\mathtt{nf}(\sigma) \preceq \mathtt{nf}(\tau)$. Therefore in order to choose in this set a canonical filter for $\sigma \preceq \tau$, it suffices to choose it for their normal forms. Hence, we define the following algorithm:

DEFINITION 3.14. *We define the ternary relation $f : \sigma \trianglelefteqslant \tau$ between a filter and two contracts in normal form by the inference*

rule

$$\frac{\mathrm{A} = \{\alpha \in (\bigcup_{\mathrm{R}\in\mathscr{R}} \mathrm{R}) \cap (\bigcup_{\mathrm{S}\in\mathscr{S}} \mathrm{S}) \mid \exists f_\alpha, f_\alpha : \sigma_\alpha \lhd \tau_\alpha\} \quad \mathscr{A} = \{\mathrm{A}' \subseteq \mathrm{A} \mid \forall \mathrm{S} \in \mathscr{S}, \mathrm{S} \cap \mathrm{A}' \in \mathscr{R}\} \quad \mathscr{A} \neq \emptyset}{\bigvee_{\mathrm{A}'\in\mathscr{A}} \coprod_{\alpha\in\mathrm{A}'} \alpha.f_\alpha : \bigoplus_{\mathrm{R}\in\mathscr{R}} \sum_{\alpha\in\mathrm{R}} \alpha.\sigma_\alpha \lhd \bigoplus_{\mathrm{S}\in\mathscr{S}} \sum_{\alpha\in\mathrm{S}} \alpha.\tau_\alpha}$$

We then extend the relation to arbitrary contracts by the following definition:

$$f : \sigma \lhd \tau \stackrel{def}{\iff} f : \mathtt{nf}(\sigma) \lhd \mathtt{nf}(\tau)\,.$$

Although it is not immediate, the definition above describes an algorithm to check whether two contracts are in relation: first the two contracts are put in normal form; then for every action α that can be immediately emitted by both normal forms, the algorithm is recursively called on the two continuations of the action. The set A represents the largest set of actions leading to continuations which are in the relation and the recursion basis occurs when $\mathrm{A} = \emptyset$. The set $\mathscr{A}$ contains the subsets $\mathrm{A}' \subseteq \mathrm{A}$ such that, by restricting each ready set of the larger contract to the actions in A', this is a ready set of the smaller contract (recall that for any two contracts σ and τ in normal form such that $\sigma \sqsubseteq \tau$, every ready set of τ is also a ready set of σ). If there is at least one such A' set of actions ($\mathscr{A} \neq \emptyset$), then σ and τ can be related. The filter defined in the conclusion is the disjunction of the filters corresponding to all these sets of actions: it uses Equation (8) to compute the greatest relevant filter.

3.3.3 Properties

The algorithm described in Definition 3.14 enjoys fundamental properties, namely (i) it proves only (soundness) and all (completeness) weak subcontract relations, (ii) in case of success it returns the largest relevant filter that proves the relation and (iii) it always terminates, which implies the decidability of the weak subcontract relation.

LEMMA 3.15 (FILTER RELEVANCE). *If $f : \sigma \lhd \tau$, then $f \leqslant I_\tau$.*

THEOREM 3.16 (SOUNDNESS). *If $f : \sigma \lhd \tau$ then $\sigma \sqsubseteq f(\tau)$.*

THEOREM 3.17 (COMPLETENESS). *If $\sigma \sqsubseteq g(\tau)$, then there exists a filter f such that $f : \sigma \lhd \tau$, and $f \geqslant g \wedge I_\tau$.*

COROLLARY 3.18. *If σ and τ are two contracts, there exists at most one filter f such that $f : \sigma \lhd \tau$. Furthermore, if $f : \sigma \lhd \tau$, then*

$$f = \max\{g \leqslant I_\tau \mid \sigma \sqsubseteq g(\tau)\} = \max\{g \leqslant I_\tau \mid g : \sigma \leq \tau\}\,.$$

The corollary above describes the logical interpretation of the algorithm as the result of a cut-elimination process. The "cut" in the system of Table 1 is given by the rule (TRANSITIVITY). This rule intersects filters, that is it minimises the proofs: therefore in order to eliminate cuts we have to find a proof with a maximum filter. However we have also to avoid useless applications of the (WEAKENING) rule, which instead maximises proofs: therefore we have to set an upper bound to filter maximisation, upper bound embodied by the definition of relevance (therefore it would be more precise to speak of a cut-weakening-elimination process).

PROPOSITION 3.19 (DECIDABILITY). *Given two contracts σ and τ, we can decide whether there exists a filter f such that $f : \sigma \lhd \tau$ and compute this filter.*

3.4 Language

The final step of our investigation is to relate contracts (which are behavioural types) with processes that implement clients and services. We do not consider any particular process language, nor do we require that clients and services be implemented using the same language. We just require that the observable behaviour of such language(s) be described by a labelled transition system and abstracted by a static type system, so that we can reason about their programs. More precisely we assume that a process 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 $\overline{a}$, which is meant to synchronise with the corresponding co-action in the process P is interacting with, or it can be an internal, invisible action $\boldsymbol{\tau}$ (not to be confused with τ that we used to range over contracts) that the process P executes autonomously. It is understood that the relation $\xrightarrow{\mu}$ is not necessarily deterministic. As usual, we let α range over visible actions and we write $P \xrightarrow{\mu}$ if $P \xrightarrow{\mu} P'$ for some process P'.

DEFINITION 3.20 (STRONG PROCESS COMPLIANCE). *Let $P \parallel Q \longrightarrow P' \parallel Q'$ be the least relation defined by the rules:*

$$\frac{P \xrightarrow{\boldsymbol{\tau}} P'}{P \parallel Q \longrightarrow P' \parallel Q} \qquad \frac{Q \xrightarrow{\boldsymbol{\tau}} Q'}{P \parallel Q \longrightarrow P \parallel Q'}$$

$$\frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\overline{\alpha}} Q'}{P \parallel Q \longrightarrow P' \parallel Q'}$$

We write $\Longrightarrow$ for the reflexive, transitive closure of $\longrightarrow$; we write $P \parallel Q \longrightarrow$ if $P \parallel Q \longrightarrow P' \parallel Q'$ for some P' and Q'; we write $P \parallel Q \not\longrightarrow$ if not $P \parallel Q \longrightarrow$.

The client P is strongly compliant *with the service Q, written $P \dashv Q$, if whenever $P \parallel Q \Longrightarrow P' \parallel Q' \not\longrightarrow$ we have $P \xrightarrow{\mathtt{e}}$.*

The intuition of this definition is that $P \parallel Q$ represents a client P and a service Q interacting with each other. When $P \dashv Q$ every interaction between P and Q terminates with P being able to emit $\mathtt{e}$, denoting the successful completion of P's task.

We also assume that a type system is given to check that a process P *implements* the contract σ. This is expressed by the judgement

$$\vdash P : \sigma$$

While we do not give details on the particular typing rules, we require typing and the reduction relation to satisfy some basic properties: essentially, contracts must describe the observational behaviour of processes and the reduction must decrease nondeterminism (entropy must always increase). In this respect, it makes sense to be able to apply the strong subcontract relation to client contracts too, where the action $\mathtt{e}$ is treated like any other action (recall that, according to Theorem 2.8, the relation $\sqsubseteq$ can be defined without any notion of "successful action" $\mathtt{e}$).

DEFINITION 3.21. *The type system is* consistent *if, whenever $\vdash P : \sigma$ and $P \xrightarrow{\mu} P'$, then $\vdash P' : \sigma'$ and (1) if $\mu = \boldsymbol{\tau}$, then $\sigma \sqsubseteq \sigma'$; (2) if $\mu = \alpha$, then $\sigma \stackrel{\alpha}{\longmapsto}$ and $\sigma(\alpha) \sqsubseteq \sigma'$. Also, the type system is* informative *if, whenever $\vdash P : \sigma$ and $\sigma \stackrel{\alpha}{\longmapsto}$, then $P \xrightarrow{\alpha}$.*

Intuitively, condition (1) states that a process performing internal actions can only make its contract more deterministic. Condition (2) 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. An informative type system does not deduce capabilities that a process does not have.

The soundness of a consistent and informative 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 3.22. *If $\vdash P : \rho$ and $\vdash Q : \sigma$ and $\rho \dashv \sigma$ then $P \dashv Q$.*

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 an operator

$$f[P]$$

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:

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

The introduction of filters into the process language 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 rule

$$\text{(TYPEFILTER)}\quad \frac{\vdash P : \sigma}{\vdash f[P] : f(\sigma)}$$

does not jeopardise the type system.

PROPOSITION 3.23. *A consistent and informative type system enriched with rule* (TYPEFILTER) *results in another consistent and informative type system.*

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 3.24. *If $\vdash P : \rho$, $\vdash Q : \sigma$, and $\rho \dashv f(\sigma)$, then $P \dashv f[Q]$.*

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.

Our approach reconciles two hitherto apparently incompatible requirements. On the one hand a subcontract relation must allow a service to be replaced or upgraded by offering more operations (width subtyping), longer interaction patterns (depth subtyping) and/or more deterministic ones. On the other hand this must be done without disrupting the behaviour of clients.

Filters provide the technical device that makes it possible. Although we initially defined filters essentially as technological mechanism to couple clients and services, filters turn out to have an elegant logical justification: they are explicit coercions between related contracts. Following the Curry-Howard isomorphism filters can be interpreted as proofs of a sound and complete deduction system for the subcontract relation. Such deduction system simultaneously refines and extends Hennessy's classical axiomatisation of the must testing preorder. Its algorithmic counterpart is obtained as a cut elimination process, which proves the coherence of subcontracting as a logical system. The canonical proof, the one produced by the algorithmic deduction system, is characterised in terms of an order relation on filters, and the algorithmic presentation allows us to show the decidability both of the subcontracting relation and of filter inference.

The theory of subcontracting is independent of the language used to implement services and clients. We do not rely on a particular language nor on a particular paradigm (objects, process algebrae, functions, ...). By defining some minimal requirements on the language (in a nut-shell, the observable behaviour of its programs must be faithfully captured by contracts), we establish the soundness of our contract system: clients always terminate interactions with any, possibly filtered, compliant service.

Filters thus play the double role of a proof tool and of programming glue between clients and services. As an aside it is nice to notice that filters can encode CCS and π-calculus restrictions: $(\nu a)P = f_{aP}[P]$ where

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

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 allows the client to use the service. Thus the filter must be computed by the query engine, which is why the algorithmic inference of filters is crucial for a practical application.

Actually, it is more realistic to imagine that a query will receive not one but several different answers, each one containing filters that may be unrelated one to each other. Therefore a second use of filters could be that of refining the search space, by specifying in a query a minimum acceptable filter. In this way the client could specify which of the possible behaviours of its "canonical" service are considered mandatory and not to be filtered out.

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

Recursion and higher-order: The contracts and filters we discussed in this work are finite. The next step of this research is the introduction of recursion both in contracts and, consequently, in filters. Actually, most of the proofs (which are available in the full version) use coinduction and they can be applied with minor changes to the recursive case. Also, for the time being synchronisation does not carry any information. Thus a further natural step is the introduction of higher order channels *à la* π-calculus.

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 (in order to be compliant it must accept a possible synchronisation with a). It is unclear to which extent such constructs would affect the $\preceq$ preorder over contracts.

Security: There exists a rich literature on security for CCS. We want to check whether it is possible to reuse or adapt some of the techniques already developed to state security results for Web service interactions and choreographies.

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 [27, 16]).

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 [10].

Relation with other formalisms: Finally, connection with other formalisms such as linear logic, session types, and game semantics must surely be deeply investigated. In particular, as regards the semantic aspects, it is interesting to notice that clients and services introduce a notion of orthogonality which suggests that a realisability semantics for contracts is worth to be explored.

References

[1] A. Alves, A. Arkin, S. Askary, C. Barreto, 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`.

[2] A. Banerji, C. Bartolini, D. Beringer, V. Chopella, et al. *Web Services Conversation Language (*WSCL*) 1.0*, March 2002. `http://www.w3.org/TR/2002/NOTE-wscl10-20020314`.

[3] 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`.

[4] D. Booth and C. 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`.

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

[6] M. Carbone, K. Honda, and N. Yoshida. A calculus of global interaction based on session types. *Electronic Notes in Theoretical Computer Science*, 171(3):127–151, 2007.

[7] M. Carbone, K. Honda, and N. Yoshida. Structured communication-centred programming for web services. In *16th European Symposium on Programming, ESOP 2007*, number 4421 in LNCS, pages 2–17. Springer, 2007.

[8] 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.

[9] 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.

[10] G. Castagna and A. Frisch. A gentle introduction to semantic subtyping. In Proc. of *PPDP '05* ACM Press (full version) and *ICALP '05*, LNCS n. 3580, Springer (summary), 2005. Joint ICALP-PPDP keynote talk.

[11] G. Chen. Soundness of coercion in the calculus of constructions. *Journal of Logic and Computation*, 14(3):405–427, 2004.

[12] R. Chinnici, H. Haas, A. Lewis, J.-J. 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`.

[13] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. *Web Services Description Language (*WSDL*) Version 2.0 Part 1: Core Language*, 2006. `http://www.w3.org/TR/2006/CR-wsdl20-20060327`.

[14] 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`.

[15] 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`.

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

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

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

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

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

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

[22] K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type discipline for structured communication-based programming. In *European Symposium on Programming (ESOP'98)*, number 1381 in LNCS, pages 122–138, 1998.

[23] C. Laneve and L. Padovani. The *must* preorder revisited – An algebraic theory for web services contracts. In *CONCUR '07*. LNCS, Springer, 2007.

[24] R. Milner. *A Calculus of Communicating Systems*. Springer, 1982.

[25] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theor. Comput. Sci*, 34:83–133, 1984.

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

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

[28] S. Soloviev, A. Jones, and Z. Luo. Some Algorithmic and Proof-Theoretical Aspects of Coercive Subtyping. In *TYPES' 96*. LNCS 1512, 173–196, Springer, 1996.

[29] K. Takeuchi, K. Honda, and M. Kubo. An interaction-based language and its typing system. In *Parallel Architectures and Languages Europe*, pages 398–413, 1994.