A Theory of Contracts for Web Services

Giuseppe Castagna
CNRS PPS, Université Denis Diderot, Paris, France
and
Nils Gesbert
University of Glasgow, Glasgow, Scotland
and
Luca Padovani
ISTI, Università degli Studi di Urbino, Urbino, Italy

Contracts are behavioral descriptions of Web services. We devise a theory of contracts that formalizes 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 *filter*, which is an explicit coercion preventing some possible behaviors of services and, in doing so, 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 axiomatization 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 outline the practical aspects of our theory by studying two different concrete syntaxes for contracts and applying each of them to Web services languages. We also explore implementation issues of filters and discuss the perspectives of future research this work opens.

Categories and Subject Descriptors: F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Parallelism and concurrency*; F.3.3 [**Logics and Meanings of Programs**]: Studies of Program Constructs—*Type structure*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*; H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*Theory and models, Web-based interaction*

General Terms: Languages, Standardization, Theory

Additional Key Words and Phrases: Web services, contracts, concurrency theory, CCS, must testing, type theory, subtyping, explicit coercions.

A preliminary version of this work appeared in the proceedings of POPL '08, the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages 2008.
Nils Gesbert is supported by EPSRC grant EP/F065708/1 (Engineering Foundations of Web Services: Theories and Tool Support).

1. INTRODUCTION

Web services are distributed components that clients can connect to and communicate with by means of standard communication protocols and platform-neutral message formats. Remarkably, Web services are equipped with machine-understandable descriptions of their interface. This aspect permits Web services to be discovered according to the information encoded in their interface, and provides very basic information for reusing and assembling existing software components. Among the capabilities that can be used as search keys are the operations provided by the service, the format or *schema* [Fallside and Walmsley 2004] of the exchanged messages, and the *contract* required to interact successfully with the service. By contract we mean the description of the external, observable behavior of a service.

The Web Service Description Language (WSDL) [Chinnici et al. 2007; Chinnici et al. 2007] is a standard technology for describing the interface exposed by a service. In WSDL, contracts are basically limited to one-way (asynchronous) and request/response (synchronous) interactions. The Web Service Conversation Language (WSCL) [Banerji et al. 2002] extends WSDL contracts by allowing the description of arbitrary, possibly cyclic sequences of exchanged messages between communicating parties. Other languages, such as the Web Service Business Execution Language (WS-BPEL) [Alves et al. 2007], provide even more detailed descriptions of services by defining the subprocess structure and more specific details regarding the service's internals. Such descriptions, which are excessively concrete and verbose to directly serve as interfaces, can be approximated and compared in terms of contracts.

Standard technologies are also available for building repositories of Web service descriptions [Bellwood et al. 2005], making it possible to perform queries for services according to their contract. Searching immediately calls for a notion of contract equivalence to perform service discovery in the same way as, say, type isomorphisms are used to perform library searches [Rittri 1993; Di Cosmo 1995]. Without a formal characterization of contracts, however, one is left with excessively demanding equivalences such as syntactical or structural equality. In fact, clients will be equally satisfied to interact with services that provide *more* capabilities than those actually required, so that it makes sense to relax the equivalence into a *subcontract preorder* (denoted by $\preceq$ in this paper).

Service discovery is not the only motivation for introducing a subcontract preorder. Another compelling reason is given by software maintenance and evolution. If clients will be satisfied to interact with services that provide more capabilities than those they require, then it is possible to use subcontracting to ensure that an upgrade of existing services (e.g., to add new capabilities or propose different interaction patterns) will not affect the existing clients. Similarly, the use of a subcontract preorder will benefit the development of Service Oriented Architectures where new applications are developed by reusing and assembling off-the-shelf software components: resorting to subcontracting enhances reuse while providing a formal framework to analyze compatibility of assemblages.

In this work we develop a formal theory of contracts that defines a coarse subcontract preorder. Along the lines of [Carpineti et al. 2006] we describe contracts by simple CCS-like terms built with just three operators: prefixing, denoted by a

dot, and two infix choice operators + (external choice) and $\oplus$ (internal choice). The contract $\alpha.\sigma$ describes a service that is capable of performing an action α, and then continues as σ. The contract $\sigma + \tau$ describes a service that lets the client decide whether to continue as σ or as τ. The contract $\sigma \oplus \tau$ describes a service that internally decides whether to continue as σ or as τ. Following CCS notation, actions are either write or read actions, the former ones 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 behavioral types of processes that do not manifest internal moves and the parallel structure.

Contracts are 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 synchronizes with one of them (i.e., it performs the corresponding co-action) or it terminates. The service contract allows us to determine the set of clients that *comply* with it, that is to say that will successfully terminate any session of interaction with the service.

As said before, the client will probably be satisfied to interact with services that offer more than what the current or 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 synchronize 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:

$$\overline{a} \oplus \overline{b}.c \preceq \overline{a} \quad \wedge \quad \overline{a} \preceq \overline{a} + \overline{b}.d \quad \not\Longrightarrow \quad \overline{a} \oplus \overline{b}.c \preceq \overline{a} + \overline{b}.d\,. \tag{1}$$

The problem can be solved by resorting to the theory of *explicit coercions* [Bruce and Longo 1990; Chen 2004; Soloviev et al. 1996]. The flawed assumption of the approach described so far, which is the one proposed in [Carpineti et al. 2006], 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 recognized that we are using $\preceq$ as an *inverse* subtyping relation for services.[1] If we denote by $:>$ the subtyping

[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 [De Nicola and Hennessy 1984]. This inversion corresponds to the duality between simulation and subtyping, viz. between

relation for services, then $\overline{a} \oplus \overline{b}.c \;\; :> \;\; \overline{a} + \overline{b}.d$ and so what we implicitly did is to apply subsumption [Cardelli 1988] 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, $:>$) is not transitive. It rather resides in the use of subsumption, since this corresponds to the use of *implicit* coercions. Coercions have many distinct characterizations 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" [Chen 2004]. For instance it is well known 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.[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 above 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 [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 metatheory in terms of explicit coercions. However, in the setting studied here, 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. 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 synchronization 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. $\sigma :> \sigma + \tau$) and/or longer interaction patterns (that is *depth subtyping*, e.g. $a :> a.\sigma$) and/or is more deterministic (e.g. $\sigma \oplus \tau :> \sigma$). On the other hand we want clients of the old service to seamlessly work with the new one.

The practical-oriented reader may better appreciate the relevance of the the-

observers and observed behaviors.

[2] In typed lambda calculus coercions are formally characterized by the fact that their type erasure is η-equivalent to the identity function, but in general coercions may be different from the identity function [Chen 2004].

ory by giving a plausible interpretation to the actions a, b, and c in the above examples. Consider a client process looking for services that accept payments, be them with a credit card or with a bank transfer, without a specific preference for one or the other. Suppose that `CreditCard` and `BankTransfer` are the two messages indicating the kind of payment available. Assume also that σ is the protocol following a credit card payment (the exact definition of σ is unimportant). The client would use $\texttt{BankTransfer} \oplus \texttt{CreditCard}.\sigma$ as search key in the service repository. The point here is that the client is happy to use whichever payment method is available. Hence, services offering only bank transfers are acceptable ($\texttt{BankTransfer} \oplus \texttt{CreditCard}.\sigma \preceq \texttt{BankTransfer}$), services offering only credit card payments are acceptable ($\texttt{BankTransfer} \oplus \texttt{CreditCard}.\sigma \preceq \texttt{CreditCard}.\sigma$), and services offering both kinds of payments are acceptable as well ($\texttt{BankTransfer} \oplus \texttt{CreditCard}.\sigma \preceq \texttt{BankTransfer} + \texttt{CreditCard}.\sigma$). As an aside, observe that, had the client searched with the $\texttt{BankTransfer} + \texttt{CreditCard}.\sigma$ key, it would have found only services offering *both* payment methods. The question is now whether any service offering bank transfers *and* some other capabilities—i.e., implementing $\texttt{BankTransfer} + \rho$ for some contract ρ—should be also returned by the query. The answer given by this article is a positive one, but this answer is not so straightforward as it may appear at first glance: consider a service whose contract is $\texttt{BankTransfer} + \texttt{CreditCard}.\tau$ where τ is incompatible with σ, in the sense that there are clients satisfied by σ but not by τ. The question is then whether we have $\texttt{BankTransfer} \oplus \texttt{CreditCard}.\sigma \preceq \texttt{BankTransfer} + \texttt{CreditCard}.\tau$. Clearly there is *one way* to satisfy the client, namely by choosing payment by bank transfer. However, in principle client and service may also agree on credit card payment (perhaps because the client tries that kind of payment first), with the risk of the client getting stuck later on during the interaction since σ and τ are not related. The solution we propose by this work is to admit the relation $\texttt{BankTransfer} \oplus \texttt{CreditCard}.\sigma \preceq \texttt{BankTransfer} + \texttt{CreditCard}.\tau$, because the service $\texttt{BankTransfer} + \texttt{CreditCard}.\tau$ (and, more generally, any service of the form $\texttt{BankTransfer} + \rho$) *can be made* comparable to (larger than) $\texttt{BankTransfer} \oplus \texttt{CreditCard}.\sigma$ by filtering out the `CreditCard` action. This filter acts as an explicit coercion that transforms every service that implements the contract $\texttt{BankTransfer} + \texttt{CreditCard}.\tau$ into a service that implements the (smaller) `BankTransfer` contract (this latter being comparable with $\texttt{BankTransfer} \oplus \texttt{CreditCard}.\sigma$ without the need of a filter), thus making it compatible with the issuer of the query.

Two observations to conclude this overview. First, the fact that we apply filters to services rather than to clients is just a presentational convenience: the same effect can be obtained by applying to the client the filter that blocks the corresponding co-actions. Second, filters must be finer-grained in blocking actions than restriction operators as defined for CCS or π-calculus. Restrictions in these calculi 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.

1.1 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 characterize 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 formalize 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 (§3.1). This subcontract relation induces a notion of weak compliance suggesting that non-interference of unexpected actions can be ensured by coercion functions, which we dub *filters* (§3.2). 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" (§3.3). The subcontract relation is shown to be decidable via the definition of a sound and complete algorithmic deduction system (§3.4). Next we show how our contracts are to be used to type Web services programming languages. In particular, we relate our contract language with a generic class of typed process languages and show the soundness of our theory of contracts: this is proved by showing that if a client process is weakly compliant with a service process via a given filter, then the filter ensures that the client will either synchronize infinitely many times with the service or it will successfully terminate (§4). The last part of the presentation is devoted to exploring the more practical aspects of this work. In particular, since our theory is stated for possibly infinite regular trees, we introduce two concrete syntaxes to finitely denote these trees and relate them with the preceding theoretical work (§5.1). Next we apply each syntax to one language proposed by the two major web standardization bodies (i.e., W3C and OASIS) (§5.2-5.3), we explore possible ways of implementing filters, and we outline how the theory can be directly implemented in WS-BPEL without requiring any modification of the WS-BPEL specification or of existing WS-BPEL processes (§5.4). An extensive discussion of related work (§6) and a conclusion that recaps our work and hints at possible tracks of future research (§7) close our presentation.

2. CONTRACTS

2.1 Syntax

Contracts are formally defined as possibly infinite trees that satisfy regularity (i.e., they have finitely many distinct subtrees: see, for instance, §4 of [Courcelle 1983]) and a contractivity condition.

DEFINITION 2.1 (CONTRACT). *Let $\mathscr{N}$ be a countable set of names. The set of contracts Σ is the set of possibly infinite terms coinductively generated by the*

following grammar:

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

and satisfying the following conditions:

(*1*) *contract terms are regular trees,*

(*2*) *on every infinite branch of a contract term there are infinitely many occurrences of the prefix constructor.*

In the definition $\mathbf{0}$ is the contract of services that do not perform any action while the other constructions were already explained in the Introduction (§1). We follow the standard convention of omitting trailing $\mathbf{0}$'s. We also work modulo associativity of each sum operator and by an abuse of notation we will sometimes denote them as n-ary operators. We then write $\sum_{i\in\{1,\ldots,n\}} \sigma_i$ for $\sigma_1+\sigma_2+\cdots+\sigma_n$ and $\bigoplus_{i\in\{1,\ldots,n\}} \sigma_i$ for $\sigma_1 \oplus \sigma_2 \oplus \cdots \oplus \sigma_n$. By convention we have $\sum_{i\in\emptyset} \sigma_i = \mathbf{0}$.

Infinite terms stand for recursive contracts. This kind of presentation is not customary in process calculi where finite representations of recursion (essentially, Kleene star, recursive equations, or rec-notations) are nearly always preferred. This is probably due to the fact that the intuition behind a finite representation can be more easily grasped. However working directly on infinite trees has two clear advantages. First and foremost, all results abstract away from the particular notation used to represent recursion: it is easy to transpose each result to each particular representation, while it is much more difficult to move from one representation to another. Second, working with infinite terms makes it quite straightforward to transpose the work to finite ones since it just suffices to forget that terms are infinite and no further modifications are needed; with finite representations of recursion, instead, definitions and results must be tailored to cope with infinite behavior and thus use constructions (such as environments for recursion variables, memoization environments in deductions) that are meaningless for finite terms.

Of course not every infinite term constructed by applying "$\oplus$", "$+$", and "$.$" is acceptable. We require the term (*i*) to be regular, so that it can be seen as the solution of some recursive definitions [Courcelle 1983], and (*ii*) to satisfy a fairly standard contractivity condition requiring that recursion must be guarded by an i/o operation, which rules out meaningless terms of the form $\mathsf{rec}\ x = x + x$.[3] As we explain in full details in the Conclusion (§7), regularity is the only real restriction that limits the expressive power of contracts with respect to fully-fledged variants of CCS: the inclusion of other constructs—such as parallel composition, restriction, relabeling—would not only bring no increase in expressive power, but

[3]Contractivity was introduced by Courcelle [Courcelle 1983] to rule out e.g. $\mathsf{rec}\ x = x$, which is *syntactically* meaningless because it is satisfied by every regular tree, but it was not meant to rule out expressions such as $\mathsf{rec}\ x = x + x$. The latter is syntactically meaningful since it denotes a particular regular tree but, in our context, it is *semantically* meaningless, because of the peculiar semantics of the "+" operator. Here we use *contractivity* in stricter interpretation, that is as a means for ruling out also terms that are *semantically* meaningless. From a technical viewpoint, these two conditions state that the binary relation $\triangleright$ defined by $\sigma_i \triangleright \sigma_1 + \sigma_2$ and $\sigma_i \triangleright \sigma_1 \oplus \sigma_2$ ($i = 1, 2$) is Noetherian (that is, strongly normalizing). This gives an induction principle on terms, principle that we will use without any further explicit reference to the relation $\triangleright$.

also result misplaced in our context since these constructs describe the internal service behavior in a framework in which only the external behavior matters.

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 [Chinnici et al. 2007; Chinnici et al. 2007] permits to describe and publish abstract and concrete descriptions of Web services. Such descriptions include the schema 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:

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

Intuitively, a client that is capable of invoking a service whose MEP is `inOnly` will also interact successfully with a service whose MEP is `robustInOnly` (depth subtyping). Conversely, 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` (since they are more deterministic), or even `inOnly`. Indeed, such a client must be able to handle *both* a communication that terminates *and* a `Fault` or `Out` message. 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 annotations within the WSDL interface. The Web service conversation language WSCL [Banerji et al. 2002] provides a more general specification language for describing complex *conversations* between two communicating parties, by means

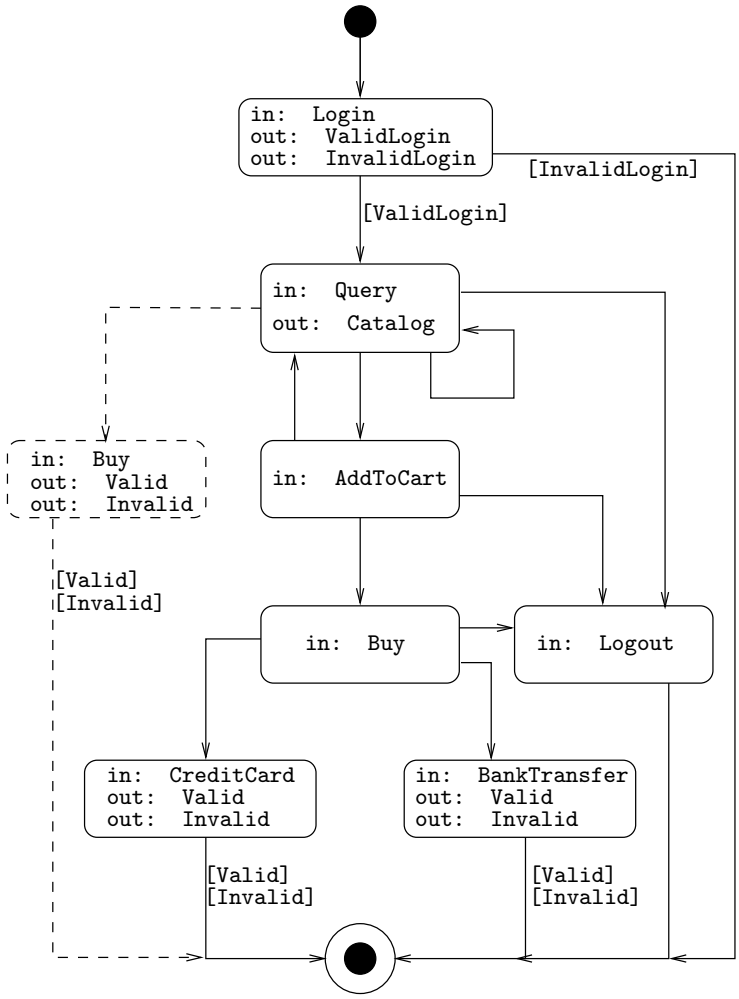

Fig. 1. Contract of an e-commerce service as a WSCL diagram.

of an activity diagram (Figure 1). The diagram is 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 of a simplified e-commerce service (Figure 1) where the client is required to login before it can select and buy items from the store. If the login is successful, the client can issue one or more queries and add items to the shopping cart. The client can buy the items in the shopping cart using one

of two payment methods, either with credit card or with a bank transfer. At any time, the client can choose to logout and leave the store. In case of purchase, the service reports whether the purchase was valid. We can represent the contract of Figure 1 (without the dashed part, which represents an extension discussed later), as the regular contract σ_1 defined by the equations:

$$\begin{aligned}
\sigma_1 &\stackrel{\text{def}}{=} \mathtt{Login}.(\overline{\mathtt{InvalidLogin}} \oplus \overline{\mathtt{ValidLogin}}.\sigma_2)\\
\sigma_2 &\stackrel{\text{def}}{=} \mathtt{Query}.\overline{\mathtt{Catalog}}.(\sigma_2 + \mathtt{Logout} + \mathtt{AddToCart}.(\sigma_2 + \mathtt{Buy}.\sigma_3))\\
\sigma_3 &\stackrel{\text{def}}{=} \mathtt{Logout} + \mathtt{CreditCard}.(\overline{\mathtt{Valid}} \oplus \overline{\mathtt{Invalid}}) + \mathtt{BankTransfer}.(\overline{\mathtt{Valid}} \oplus \overline{\mathtt{Invalid}})
\end{aligned}$$

Unlabeled transitions in Figure 1 correspond to external choices in the contract, whereas labeled transitions correspond to internal choices. The use of recursion in the definition of σ_2, corresponds to the presence of (two) cycles in the WSCL graph.

Let us recast in this setting the three forms of subtyping we described in the Introduction (§1). First, it is clear that clients compliant with the service above will always be happy with more deterministic servers that, for instance, never deny the access ($\overline{\mathtt{InvalidLogin}} \oplus \overline{\mathtt{ValidLogin}} \preceq \overline{\mathtt{ValidLogin}}$) as well as with servers that offer longer interactions, such as the fact of proposing an invoice after the payment ($\overline{\mathtt{Valid}} \preceq \overline{\mathtt{Valid}}.\overline{\mathtt{Invoice}}$). Now assume that the service is extended (by width subtyping) with "1-click ordering" capability, so that if the client has already bought items, perhaps in some previous sessions, then it is allowed to buy further items without adding them to the shopping cart and without the need to re-send the payment information (dashed part in Figure 1). The contract σ_2 would change to σ_2' as follows:

$$\begin{aligned}
\sigma_2' &\stackrel{\text{def}}{=} \mathtt{Query}.\overline{\mathtt{Catalog}}.(\sigma_2' + \mathtt{Logout} + \mathtt{AddToCart}.(\sigma_2' + \mathtt{Buy}.\sigma_3) + \sigma_4)\\
\sigma_4 &\stackrel{\text{def}}{=} \mathtt{Buy}.(\overline{\mathtt{Valid}} \oplus \overline{\mathtt{Invalid}})
\end{aligned}$$

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, the transitivity problem we pointed out in the Introduction (§1) might arise. Indeed, assume that, before the extension, we have a client trying to send a `Buy` message right after receiving a catalog from the service (e.g., because it was written in a generic way so as to be able to interact with some different service in which the `Buy` operation is used) and that after sending such a message it behaves according to some contract ρ_B. Assume also that this client is compliant with the former service for the simple reason that, since the former service did not provide a "1-click ordering" capability, whatever contract ρ_B the client provided after the `Buy` action was irrelevant to establish compliance. If we extend the services by "1-click ordering" capability without using filters, then this client may result no longer compatible with the service, specifically if ρ_B is incompatible with the continuation of σ_4 after `Buy`. In order to avoid that the extension of the service disrupts compatibility with existing clients, the "1-click ordering" extension must resort to a filter that will block any `Buy` message that immediately follows a `Catalog` message (see §3.2.1 for a precise definition of the filter and Remark 5.5 for a finer analysis of this example).

2.3 Semantics

Contracts describe the behavior of the processes that implement them. This behavior is determined by the actions that are offered by a process and the way in which they are offered (note that both $\sigma \oplus \tau$ and $\sigma + \tau$ offer the same actions). This is formally stated by the Definitions 2.2 and 2.4 given below.

DEFINITION 2.2 (TRANSITION). *Let $\sigma \overset{\alpha}{\mapsto\!\!\!\!\!\!/\;\;}$ be the least relation such that:*

$$\mathbf{0} \overset{\alpha}{\mapsto\!\!\!\!\!\!/\;\;} \qquad \frac{\alpha \neq \beta}{\beta.\sigma \overset{\alpha}{\mapsto\!\!\!\!\!\!/\;\;}} \qquad \frac{\sigma \overset{\alpha}{\mapsto\!\!\!\!\!\!/\;\;} \quad \tau \overset{\alpha}{\mapsto\!\!\!\!\!\!/\;\;}}{\sigma \oplus \tau \overset{\alpha}{\mapsto\!\!\!\!\!\!/\;\;}} \qquad \frac{\sigma \overset{\alpha}{\mapsto\!\!\!\!\!\!/\;\;} \quad \tau \overset{\alpha}{\mapsto\!\!\!\!\!\!/\;\;}}{\sigma + \tau \overset{\alpha}{\mapsto\!\!\!\!\!\!/\;\;}}$$

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

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

$$\frac{\sigma \overset{\alpha}{\longmapsto} \sigma' \quad \tau \overset{\alpha}{\longmapsto} \tau'}{\sigma + \tau \overset{\alpha}{\longmapsto} \sigma' \oplus \tau'} \qquad \frac{\sigma \overset{\alpha}{\longmapsto} \sigma' \quad \tau \overset{\alpha}{\mapsto\!\!\!\!\!\!/\;\;}}{\sigma + \tau \overset{\alpha}{\longmapsto} \sigma'} \qquad \frac{\sigma \overset{\alpha}{\mapsto\!\!\!\!\!\!/\;\;} \quad \tau \overset{\alpha}{\longmapsto} \tau'}{\sigma + \tau \overset{\alpha}{\longmapsto} \tau'}$$

$$\frac{\sigma \overset{\alpha}{\longmapsto} \sigma' \quad \tau \overset{\alpha}{\longmapsto} \tau'}{\sigma \oplus \tau \overset{\alpha}{\longmapsto} \sigma' \oplus \tau'} \qquad \frac{\sigma \overset{\alpha}{\longmapsto} \sigma' \quad \tau \overset{\alpha}{\mapsto\!\!\!\!\!\!/\;\;}}{\sigma \oplus \tau \overset{\alpha}{\longmapsto} \sigma'} \qquad \frac{\sigma \overset{\alpha}{\mapsto\!\!\!\!\!\!/\;\;} \quad \tau \overset{\alpha}{\longmapsto} \tau'}{\sigma \oplus \tau \overset{\alpha}{\longmapsto} \tau'}$$

We write $\sigma \overset{\alpha}{\longmapsto}$ if there exists σ' such that $\sigma \overset{\alpha}{\longmapsto} \sigma'$.

The relation $\overset{\alpha}{\longmapsto}$ is different from standard transition relations for CCS processes [Milner 1982]. For example, there is always at most one contract σ' such that $\sigma \overset{\alpha}{\longmapsto} \sigma'$, 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 an external communicating party*. Thus $a.b + a.c \overset{a}{\longmapsto} b \oplus c$ 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* (see the section about related work (§6) for a detailed discussion about this difference in perspective).

NOTATION 2.3. *We use* $\mathsf{init}(\sigma)$ *to denote the set of actions that can be immediately emitted by σ, that is* $\mathsf{init}(\sigma) = \{\alpha \mid \sigma \overset{\alpha}{\longmapsto}\}$.

Let $\sigma \overset{\alpha}{\longmapsto}$. We write $\sigma(\alpha)$ for the unique continuation of σ after α, that is, the contract σ' such that $\sigma \overset{\alpha}{\longmapsto} \sigma'$. We extend the notion of continuation to sequences of actions. Let φ denote a possibly empty, finite string of actions. If $\varphi = \varepsilon$, then $\sigma \overset{\varphi}{\longmapsto} \sigma$ and we have $\sigma(\varphi) = \sigma$; if $\varphi = \alpha\varphi'$, then $\sigma \overset{\varphi}{\longmapsto} \sigma'$ if $\sigma \overset{\alpha}{\longmapsto} \sigma' \overset{\varphi'}{\longmapsto} \sigma''$ and we have $\sigma(\varphi) = \sigma''$.

The labeled 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 2.4 (observable ready sets). *Let $\mathscr{P}_f(\mathscr{N} \cup \overline{\mathscr{N}})$ be the set of finite subsets of $\mathscr{N} \cup \overline{\mathscr{N}}$, called* ready sets. *Let also $\sigma \Downarrow \mathrm{R}$ be the least relation between contracts σ in Σ and ready sets R in $\mathscr{P}_f(\mathscr{N} \cup \overline{\mathscr{N}})$ such that:*

$$\mathbf{0} \Downarrow \emptyset \qquad \alpha.\sigma \Downarrow \{\alpha\} \qquad \frac{\sigma \Downarrow \mathrm{R} \quad \tau \Downarrow \mathrm{S}}{\sigma + \tau \Downarrow \mathrm{R} \cup \mathrm{S}} \qquad \frac{\sigma \Downarrow \mathrm{R}}{\sigma \oplus \tau \Downarrow \mathrm{R}} \qquad \frac{\tau \Downarrow \mathrm{R}}{\sigma \oplus \tau \Downarrow \mathrm{R}}$$

Notation 2.5. *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 $\mathsf{co}(\mathrm{R}) = \{\overline{\alpha} \mid \alpha \in \mathrm{R}\}$.*

2.4 The problem

We now possess all the technical instruments to formally state the problem we described in the Introduction (§1) and recalled at the end of §2.2. This first requires the precise definition of *compliance*. Recall that, intuitively, the behavior of a client complies with the behavior of a service if for every set of actions that the service may offer, the client either synchronizes with one of them, or it terminates successfully. The behavior of clients, as well as that of services, is described by contracts. Therefore we need to define when a contract ρ describing the behavior of a client complies with a contract σ describing the behavior of a service. For this we reserve a special action 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 is in a state where this action is available.

Definition 2.6 (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 $\mathsf{e} \in \mathrm{R}$ or $\mathsf{co}(\mathrm{R}) \cap \mathrm{S} \neq \emptyset$, and

(2) $\rho \overset{\overline{\alpha}}{\longmapsto} \rho'$ and $\sigma \overset{\alpha}{\longmapsto} \sigma'$ 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 synchronize with an action among those offered by the service ($\mathsf{co}(\mathrm{R}) \cap \mathrm{S} \neq \emptyset$) or the client terminates successfully ($\mathsf{e} \in \mathrm{R}$), and (2) whenever a synchronization 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 [De Nicola and Hennessy 1984].

Definition 2.7 (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 2.8 (COINDUCTIVE STRONG SUBCONTRACT). *$\mathscr{S}$ 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 2.9. *$\sqsubseteq$ is the largest coinductive strong subcontract relation.*

PROOF. First of all we prove that $\sqsubseteq$ is a coinductive subcontract relation. Assume $\sigma\sqsubseteq\tau$. As regards condition (1) in the definition of coinductive strong subcontract relation, let $\mathrm{R}_1,\ldots,\mathrm{R}_n$ be the ready sets of σ. By contradiction, assume that there exists R' such that $\tau\Downarrow\mathrm{R}'$ and for every $1\leq i\leq n$ there exists $\alpha_i\in\mathrm{R}_i$ such that $\alpha_i\notin\mathrm{R}'$. Let $\rho\stackrel{\text{def}}{=}\sum_{1\leq i\leq n}\overline{\alpha_i}.\mathsf{e}$. Then $\rho\dashv\sigma$ but $\rho\not\dashv\tau$, which is not possible. Hence condition (1) is satisfied. As regards condition (2) in the definition of coinductive strong subcontract relation, assume $\tau\stackrel{\alpha}{\longmapsto}$. By contradiction, assume $\sigma\stackrel{\alpha}{\not\longmapsto}$. Then $\mathsf{e}+\overline{\alpha}\dashv\sigma$ but $\mathsf{e}+\overline{\alpha}\not\dashv\tau$, which is not possible. Hence $\sigma\stackrel{\alpha}{\longmapsto}$. Now we have to prove that $\sigma(\alpha)\sqsubseteq\tau(\alpha)$. Let ρ' be such that $\rho'\dashv\sigma(\alpha)$. Then $\mathsf{e}+\overline{\alpha}.\rho'\dashv\sigma$ hence $\mathsf{e}+\overline{\alpha}.\rho'\dashv\tau$, thus $\rho'\dashv\tau(\alpha)$ by definition of strong compliance. Hence $\sigma(\alpha)\sqsubseteq\tau(\alpha)$ because ρ' is arbitrary.

Now we prove that $\sqsubseteq$ is indeed the largest coinductive subcontract relation, namely that every coinductive subcontract relation is included in $\sqsubseteq$. Let $\mathscr{S}$ be a coinductive strong subcontract relation and let $(\sigma,\tau)\in\mathscr{S}$. Let $\rho\dashv\sigma$, then there exists a strong compliance relation $\mathscr{C}$ such that $(\rho,\sigma)\in\mathscr{C}$. To get $\rho\dashv\tau$ it is sufficient to prove that

$$\mathscr{C}'\stackrel{\text{def}}{=}\{(\rho',\tau')\mid\exists\sigma',(\rho',\sigma')\in\mathscr{C}\wedge(\sigma',\tau')\in\mathscr{S}\}$$

is a strong compliance relation, since $(\rho,\tau)\in\mathscr{C}'$. Let $(\rho',\tau')\in\mathscr{C}'$ and let σ' be the corresponding contract given by the definition of $\mathscr{C}'$. As regards condition (1) in Definition 2.6, let $\rho'\Downarrow\mathrm{R}$ and $\tau'\Downarrow\mathrm{S}$. If $\mathsf{e}\in\mathrm{R}$ there is nothing to prove. Assume $\mathsf{e}\notin\mathrm{R}$. From $(\sigma',\tau')\in\mathscr{S}$ there exists $\mathrm{S}'\subseteq\mathrm{S}$ such that $\sigma'\Downarrow\mathrm{S}'$. From $(\rho',\sigma')\in\mathscr{C}$ we know $\mathsf{co}(\mathrm{R})\cap\mathrm{S}'\neq\emptyset$, hence we conclude $\mathsf{co}(\mathrm{R})\cap\mathrm{S}\neq\emptyset$. As regards condition (2) in Definition 2.6, assume $\rho'\stackrel{\overline{\alpha}}{\longmapsto}$ and $\tau'\stackrel{\alpha}{\longmapsto}$. From $(\sigma',\tau')\in\mathscr{S}$ we know that $\sigma'\stackrel{\alpha}{\longmapsto}$ and $(\sigma'(\alpha),\tau'(\alpha))\in\mathscr{S}$. From $(\rho',\sigma')\in\mathscr{C}$ we know that $(\rho'(\alpha),\sigma'(\alpha))\in\mathscr{C}$, hence we conclude $(\rho'(\alpha),\tau'(\alpha))\in\mathscr{C}'$ by definition of $\mathscr{C}'$. □

It turns out that the relation $\sqsubseteq$ is the *must testing preorder* as defined in [De Nicola and Hennessy 1984; 1987]. 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 the relation $\preceq$ that we informally defined in the Introduction (§1), holds. However $\sqsubseteq$ is stronger than $\preceq$ since, for example, $\overline{a}\not\sqsubseteq\overline{a}+\overline{b}$. Indeed $a.\mathsf{e}+b\dashv\overline{a}$

but $a.\mathtt{e} + b \not\dashv \overline{a} + \overline{b}$. In general, the must preorder allows neither width nor depth extensions of contracts. It should also be noted that, while $\sqsubseteq$ coincides with the *must* testing preorder, the notion of compliance that we use differs significantly from the notion of "passing a test" in the classical testing framework. For example, on the one hand we have $\mathtt{e} + b.c.\mathtt{e} \not\dashv \overline{b}.\overline{d}$ because the client $\mathtt{e} + b.c.\mathtt{e}$ gets stuck if it synchronizes on b. On the other hand, the process $\overline{b}.\overline{d}$ "passes the test" $\mathtt{e} + b.c.\mathtt{e}$ because, at some point during the interaction, the test/client can emit $\mathtt{e}$. In the context of Web services, the fact that a client may successfully terminate at some stage of the interaction does not automatically imply that the client will eventually terminate successfully regardless of the synchronizations occurring on later stages. In summary, the fact that $\sqsubseteq$ coincides with the *must* testing preorder may seem obvious as both relations deal with reduction of nondeterminism, but this coincidence is not at all obvious *a priori*. A more detailed discussion of these aspect, also taking into account diverging behaviors, can be found in [Laneve and Padovani 2007].

In previous work [Carpineti et al. 2006] 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 3.14 and then apply the transformation described above; see [Carpineti et al. 2006] for details). Intuitively $\mathtt{dual}(\sigma)$ denotes the contract of a "canonical" client complying with σ services. Then using this intuition one can informally define a new relation on service contracts as:

$$\sigma \ltimes \tau \stackrel{\text{def}}{\iff} \mathtt{dual}(\sigma) \dashv \tau \tag{2}$$

In words, a contract σ is a subcontract of τ if and only if its canonical client complies with τ (see the proof of Theorem 3.4 for a formal definition of $\ltimes$ and a more precise characterization of $\mathtt{dual}$).

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: $\overline{a}.\mathtt{e} + \overline{b}.\overline{c}.\mathtt{e} \not\dashv a + b.d$ 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 synchronization on the name b happens. While contracts deal with non-determinism that is internal to each process—be it a client or a service—, they cannot handle the "system" non-determinism that springs from process synchronization. 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 synchronization on the name b. By preventing the synchronization 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 synchronization. Our theory of contracts will define a safe substitutability relation for services that *can be made* compatible.

3. 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 clients of σ servers 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 behaviors of the τ contract that do not belong to the possible behaviors of σ world, and leave the others unchanged. This is, precisely, the characterization of an explicit coercion from τ to σ (recall that the subcontract relation is the inverse of a service subtyping relation; *cf.* Footnote 1 page 4): an embedding function that maps possible behaviors of τ into the same behaviors 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 formalized by the following definition:

DEFINITION 3.1 (WEAK SUBCONTRACT). *$\mathscr{W}$ is a* weak subcontract relation *if $(\sigma, \tau) \in \mathscr{W}$ implies that if $\tau \Downarrow \mathrm{R}$, then there exists $\mathrm{S_R} \subseteq \mathrm{R}$ such that (1) $\sigma \Downarrow \mathrm{S_R}$ 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 essentially 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 behaviors (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 characterization 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 $\mathsf{e}\in\mathrm{R}$ 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.

The existence of the set A in the above definition is *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 $\rho\dashv\!\!\!\mid\tau$.*

PROOF. ($\Rightarrow$) Let $\mathscr{W}$ be a weak subcontract relation such that $(\sigma,\tau)\in\mathscr{W}$ and assume $\rho\dashv\!\!\!\mid\sigma$. Let $\mathscr{D}$ be a weak compliance relation such that $(\rho,\sigma)\in\mathscr{D}$. To get $\rho\dashv\!\!\!\mid\tau$ it suffices to prove that

$$\mathscr{D}'\overset{\text{def}}{=}\{(\rho',\tau')\mid\exists\sigma',(\rho',\sigma')\in\mathscr{D}\wedge(\sigma',\tau')\in\mathscr{W}\}$$

is a weak compliance relation since $(\rho,\tau)\in\mathscr{D}'$. Let $(\rho',\tau')\in\mathscr{D}'$, and let σ' be the corresponding contract given by the definition of $\mathscr{D}'$. Let A' be the set of actions given by $(\rho',\sigma')\in\mathscr{D}$. Let $\mathrm{S}_1,\ldots,\mathrm{S}_n$ be the ready sets of τ'. Because $(\sigma',\tau')\in\mathscr{W}$, for each S_i there exists a ready set $\mathrm{S}'_i\subseteq\mathrm{S}_i$ of σ' which satisfies the conditions of Definition 3.1. Let $\mathrm{A}\overset{\text{def}}{=}\mathrm{A}'\cap\bigcup_{i=1}^{n}\mathrm{S}'_i$. We now prove that this A satisfies the conditions of Definition 3.2 for (ρ',τ'). As regards condition (1), assume $\rho'\Downarrow\mathrm{R}$ and $\tau'\Downarrow\mathrm{S}$. Then $\mathrm{S}=\mathrm{S}_i$ for some i. If $\mathsf{e}\in\mathrm{R}$ there is nothing to prove. Assume $\mathsf{e}\notin\mathrm{R}$. Then from $(\rho',\sigma')\in\mathscr{D}$ we know that $\mathtt{co}(\mathrm{R})\cap\mathrm{A}'\cap\mathrm{S}'_i\neq\emptyset$. We have $\mathrm{A}'\cap\mathrm{S}'_i=\mathrm{A}\cap\mathrm{S}'_i\subseteq\mathrm{A}\cap\mathrm{S}_i$, hence we conclude $\mathtt{co}(\mathrm{R})\cap\mathrm{A}\cap\mathrm{S}_i\neq\emptyset$. As regards condition (2), assume $\alpha\in\mathrm{A}$ and $\rho'\overset{\overline{\alpha}}{\longmapsto}$ and $\tau'\overset{\alpha}{\longmapsto}$. Then $\sigma'\overset{\alpha}{\longmapsto}$, because α is in some S'_i; and α is also in A', thus $(\rho'(\alpha),\sigma'(\alpha))\in\mathscr{D}$ by definition of A'. From $\alpha\in\mathrm{S}'_i$ we also have that $(\sigma'(\alpha),\tau'(\alpha))\in\mathscr{W}$. We conclude $(\rho'(\alpha),\tau'(\alpha))\in\mathscr{D}'$ by definition of $\mathscr{D}'$.

($\Leftarrow$) We prove that

$$\mathscr{W}\overset{\text{def}}{=}\{(\sigma,\tau)\mid\forall\rho,\rho\dashv\!\!\!\mid\sigma\Rightarrow\rho\dashv\!\!\!\mid\tau\}$$

is a weak subcontract relation. Let $(\sigma,\tau)\in\mathscr{W}$. As regards condition (1) in Definition 3.1, let $\mathrm{R}_1,\ldots,\mathrm{R}_n$ be all the (distinct) ready sets of σ. By contradiction, suppose that there exists a ready set R' such that $\tau\Downarrow\mathrm{R}'$ and for every $1\leq i\leq n$ we have $\mathrm{R}_i\not\subseteq\mathrm{R}'$, namely there exists $\alpha_i\in\mathrm{R}_i\setminus\mathrm{R}'$. Let $\rho\overset{\text{def}}{=}\sum_{1\leq i\leq n}\overline{\alpha_i}.\mathsf{e}$. By construction we have $\rho\dashv\!\!\!\mid\sigma$ but $\rho\not\dashv\!\!\!\mid\tau$, which is not possible. As regards condition (2) in Definition 3.1, let $\tau\Downarrow\mathrm{R}'$ and $k\in\{1,\ldots,n\}$ be such that $\mathrm{R}_k\subseteq\mathrm{R}'$ and R_k is *minimal* among the R_i's. We take R_k as the ready set $\mathrm{S}_{\mathrm{R}'}$ in the definition of weak subcontract relation. If $\mathrm{R}_k=\emptyset$, then condition (2) trivially holds. Assume $\mathrm{R}_k\neq\emptyset$. For every $\alpha\in\mathrm{R}_k$, let ρ_α be a client contract such that $\rho_\alpha\dashv\!\!\!\mid\sigma(\alpha)$. Notice that for

every $i \in \{1, \ldots, n\} \setminus \{k\}$, we have $\mathrm{R}_i \setminus \mathrm{R}_k \neq \emptyset$ because the R_i's are all distinct and R_k is minimal. Let

$$\rho \stackrel{\text{def}}{=} \sum_{i \in \{1,\ldots,n\} \setminus \{k\}, \beta \in \mathrm{R}_i \setminus \mathrm{R}_k} \overline{\beta}.\mathsf{e} + \bigoplus_{\alpha \in \mathrm{R}_k} \overline{\alpha}.\rho_\alpha \,.$$

By construction $\rho \dashv\!\shortmid \sigma$, hence $\rho \dashv\!\shortmid \tau$ by definition of $\mathscr{W}$. Furthermore, the set A in $\rho \dashv\!\shortmid \sigma$ must be at least as large as R_k because, by construction of ρ, ρ cannot be (weakly) compliant with σ if any of the actions in R_k is filtered out. Thus, for every $\alpha \in \mathrm{R}_k$, from $\rho \dashv\!\shortmid \sigma$ we derive $\rho_\alpha \dashv\!\shortmid \sigma(\alpha)$, hence $\rho_\alpha \dashv\!\shortmid \tau(\alpha)$. Because the ρ_α's are arbitrary, we conclude $(\sigma(\alpha), \tau(\alpha)) \in \mathscr{W}$ by definition of $\mathscr{W}$. □

3.1.2 *Comparison with other relations.* In §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 3.4. *The subcontract relation $\preceq$ is the transitive closure of $\ltimes$.*

PROOF. We did not define $\mathtt{dual}(\sigma)$ formally here, so we will give an equivalent definition of $\ltimes$ not based on the notion of dual contract, which was also the definition used in [Carpineti et al. 2006], and just give the intuition of how we obtain it using $\mathtt{dual}(\sigma)$. The important property about $\mathtt{dual}(\sigma)$ is that its ready sets are defined as all the possible sets obtained by picking one action in each ready set of σ, and taking their co-actions. 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 2.6 and assume $(\mathtt{dual}(\sigma), \tau) \in \mathscr{C}$ where $\mathscr{C}$ is a strong compliance relation, then the first condition says that any ready set of τ contains at least one action from each ready set of $\mathtt{dual}(\sigma)$, which is equivalent to the fact that it contains a ready set of σ. Translation of condition (2) is straightforward, so we get that $\ltimes$ is the largest relation $\mathscr{R}$ such that $(\sigma, \tau) \in \mathscr{R}$ implies:

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

Now let us prove that $\preceq$ is the transitive closure of the relation thus defined. Note that the condition (1) is the same in both relations, and that condition (2) in Definition 3.1 is a weakened version of condition (1) for $\mathscr{R}$, so obviously $\ltimes \subseteq \preceq$ and so does the transitive closure of $\ltimes$, $\preceq$ being itself transitive. So what we have to show is that two contracts related by $\preceq$ are also related by the transitive closure of $\ltimes$. Let $\mathscr{W}$ be a weak subcontract relation such that $(\sigma, \tau) \in \mathscr{W}$. Let

$$\begin{aligned} \mathscr{R}_1 &\stackrel{\text{def}}{=} \{(\sigma, \textstyle\bigoplus_{\tau \Downarrow \mathrm{R}} \sum_{\alpha \in \mathrm{S}_\mathrm{R}} \alpha.\sigma(\alpha)) \mid (\sigma, \tau) \in \mathscr{W}\} \\ \mathscr{R}_2 &\stackrel{\text{def}}{=} \{(\textstyle\bigoplus_{\tau \Downarrow \mathrm{R}} \sum_{\alpha \in \mathrm{S}_\mathrm{R}} \alpha.\sigma(\alpha), \tau) \mid (\sigma, \tau) \in \mathscr{W}\} \end{aligned}$$

where, for each ready set R of τ, we write S_R for the ready set of σ such that $\mathrm{S}_\mathrm{R} \subseteq \mathrm{R}$ that satisfies condition (2) in Definition 3.1. It is trivial to verify that $\mathscr{R}_1 \cup \mathscr{R}_2 \subseteq \ltimes$, from which we conclude that $\mathscr{W}$ is included in 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 2.8, 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.

3.2 Filters

A filter is the specification of a (possibly infinite) prefix-closed regular set of *traces*:

DEFINITION 3.5 (FILTERS). *A* filter *is a possibly infinite term coinductively generated by the following grammar.*

$$f ::= \mathbf{0} \mid \alpha.f \mid f \vee f \mid f \wedge f$$

and satisfying the following conditions:

(*1*) *filter terms are regular trees,*

(*2*) *on every infinite branch of a filter term there are infinitely many occurrences of the prefix constructor.*

The filter $\mathbf{0}$ is the one that allows no actions; the filter $\alpha.f$ allows those traces beginning with action α and followed by the traces allowed by f; the filter $f \vee g$ represents the *disjunction* of f and g and it allows those traces that are allowed either by f or by g; finally, the filter $f \wedge g$ represents the *conjunction* of f and g and it allows those traces that are allowed by both f and g. The conditions of regularity and contractivity are standard. The latter also provides a well-founded order for the induction (*cf.* Footnote 3) used in the next definitions.

Much like contracts, filters too are equipped with the relations $\overset{\alpha}{\mapsto\!\!\!\!\!/}$ and $\overset{\alpha}{\longmapsto}$.

DEFINITION 3.6 (FILTER TRANSITION). *Let $f \overset{\alpha}{\mapsto\!\!\!\!\!/}$ be the least relation such that:*

$$\mathbf{0} \overset{\alpha}{\mapsto\!\!\!\!\!/} \qquad \frac{\alpha \neq \beta}{\beta.f \overset{\alpha}{\mapsto\!\!\!\!\!/}} \qquad \frac{f \overset{\alpha}{\mapsto\!\!\!\!\!/} \quad g \overset{\alpha}{\mapsto\!\!\!\!\!/}}{f \vee g \overset{\alpha}{\mapsto\!\!\!\!\!/}} \qquad \frac{f \overset{\alpha}{\mapsto\!\!\!\!\!/}}{f \wedge g \overset{\alpha}{\mapsto\!\!\!\!\!/}} \qquad \frac{g \overset{\alpha}{\mapsto\!\!\!\!\!/}}{f \wedge g \overset{\alpha}{\mapsto\!\!\!\!\!/}}$$

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

$$\alpha.f \overset{\alpha}{\longmapsto} f \qquad \frac{f \overset{\alpha}{\longmapsto} f' \quad g \overset{\alpha}{\longmapsto} g'}{f \vee g \overset{\alpha}{\longmapsto} f' \vee g'} \qquad \frac{f \overset{\alpha}{\longmapsto} f' \quad g \overset{\alpha}{\mapsto\!\!\!\!\!/}}{f \vee g \overset{\alpha}{\longmapsto} f'} \qquad \frac{f \overset{\alpha}{\longmapsto} f' \quad g \overset{\alpha}{\longmapsto} g'}{f \wedge g \overset{\alpha}{\longmapsto} f' \wedge g'}$$

and closed under mirror cases for the disjunction. We write $f \overset{\alpha}{\longmapsto}$ if there exists f' such that $f \overset{\alpha}{\longmapsto} f'$.

This transition relation allows us to state more formally what we said above about sets of traces: the semantics of a filter is the prefix-closed regular language defined on the alphabet of actions by $[\![f]\!] \overset{\text{def}}{=} \{\varepsilon\} \cup \{\alpha\varphi \mid f \overset{\alpha}{\longmapsto} f', \varphi \in [\![f']\!]\}$. Then it can be easily checked that $[\![f \vee g]\!] = [\![f]\!] \cup [\![g]\!]$ and $[\![f \wedge g]\!] = [\![f]\!] \cap [\![g]\!]$ (notice that the intersection and the union of prefix-closed sets is again prefix-closed).

We consider two filters to be equal if they have the same semantics and adopt a notation similar to the one for contracts: we write $\bigvee_{i \in \{1,\ldots,n\}} f_i$ for $f_1 \vee f_2 \vee \cdots \vee f_n$

and $\bigwedge_{i\in\{1,\ldots,n\}} f_i$ for $f_1 \wedge f_2 \wedge \cdots \wedge f_n$. By convention we have $\bigvee_{i\in\emptyset} f_i = \mathbf{0}$. The application of a filter f to a contract σ, written $f(\sigma)$, produces another contract where only the allowed actions are visible:

DEFINITION 3.7 (FILTER APPLICATION). *The application of a filter f to a contract σ, written $f(\sigma)$, is inductively defined as follows:*

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

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 3.8. *$\sigma \sqsubseteq \tau$ implies $f(\sigma) \sqsubseteq f(\tau)$.*

PROOF. It is sufficient to show that

$$\mathscr{S} \overset{\text{def}}{=} \{(f(\sigma), f(\tau)) \mid \sigma \sqsubseteq \tau\}$$

is a strong subcontract relation. Let $(f(\sigma), f(\tau)) \in \mathscr{S}$. As regards condition (1) in the definition of strong subcontract relation, assume $f(\tau) \Downarrow \mathrm{S}$. Then there exists S' such that $\tau \Downarrow \mathrm{S}'$ and $\mathrm{S} = f(\mathrm{S}')$. From $\sigma \sqsubseteq \tau$ we derive that there exists R such that $\sigma \Downarrow \mathrm{R}$ and $\mathrm{R} \subseteq \mathrm{S}'$. We observe $f(\mathrm{R}) \subseteq f(\mathrm{S}') = \mathrm{S}$ and we conclude by observing that $f(\sigma) \Downarrow f(\mathrm{R})$. As regards condition (2), assume $f(\tau) \overset{\alpha}{\longmapsto}$. Then $f \overset{\alpha}{\longmapsto} f'$ and $\tau \overset{\alpha}{\longmapsto}$. From the hypothesis $\sigma \sqsubseteq \tau$ we derive $\sigma \overset{\alpha}{\longmapsto}$ and $\sigma(\alpha) \sqsubseteq \tau(\alpha)$. We conclude $f(\sigma) \overset{\alpha}{\longmapsto}$ and $(f(\sigma)(\alpha), f(\tau)(\alpha)) \in \mathscr{S}$ because $f(\sigma)(\alpha) = f'(\sigma(\alpha))$ and $f(\tau)(\alpha) = f'(\tau(\alpha))$. □

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

THEOREM 3.9. *$\sigma \preceq \tau$ if and only if there exists a filter f such that $\sigma \sqsubseteq f(\tau)$.*

PROOF. With an abuse of notation we write $f(\mathrm{R})$, the application of a filter f to a set of actions R, for the set $\{\alpha \in \mathrm{R} \mid f \overset{\alpha}{\longmapsto}\}$.

($\Leftarrow$) Let $\mathscr{S}$ be a coinductive strong subcontract relation. We show that

$$\mathscr{W} \overset{\text{def}}{=} \{(\sigma, \tau) \mid \exists f, (\sigma, f(\tau)) \in \mathscr{S}\}$$

is a weak subcontract relation. Let $(\sigma, \tau) \in \mathscr{W}$ and let f be the corresponding filter. Regarding condition (1) in Definition 3.1, assume $\tau \Downarrow \mathrm{R}$. From $(\sigma, f(\tau)) \in \mathscr{S}$ we know that there exists $\mathrm{S} \subseteq f(\mathrm{R})$ such that $\sigma \Downarrow \mathrm{S}$ and we conclude $\mathrm{S} \subseteq f(\mathrm{R}) \subseteq \mathrm{R}$. Regarding condition (2) in Definition 3.1, take $\alpha \in \mathrm{S}$. From $(\sigma, f(\tau)) \in \mathscr{S}$ we know $(\sigma(\alpha), f_\alpha(\tau(\alpha))) \in \mathscr{S}$ where $f \overset{\alpha}{\longmapsto} f_\alpha$. Hence we conclude $(\sigma(\alpha), \tau(\alpha)) \in \mathscr{W}$.

($\Rightarrow$) Let $\mathscr{W}$ be a weak subcontract relation. For every $(\sigma, \tau) \in \mathscr{W}$, let

$$\mathrm{A}(\sigma, \tau) \overset{\text{def}}{=} \bigcup_{\tau \Downarrow \mathrm{R}} \mathrm{S}_\mathrm{R}$$

where $\mathrm{S_R} \subseteq \mathrm{R}$ is such that $\sigma \Downarrow \mathrm{S_R}$ and $\mathrm{S_R}$ satisfies condition (2) in Definition 3.1. Basically $\mathrm{A}(\sigma,\tau)$ is the set of actions that need not be shielded for proving that $\sigma \preceq \tau$. Notice that $\alpha \in \mathrm{A}(\sigma,\tau)$ implies $\sigma \stackrel{\alpha}{\longmapsto}$ and $\tau \stackrel{\alpha}{\longmapsto}$.

For every $(\sigma,\tau) \in \mathscr{W}$, let

$$f_{(\sigma,\tau)} \stackrel{\text{def}}{=} \bigvee_{\alpha \in \mathrm{A}(\sigma,\tau)} \alpha.f_{(\sigma(\alpha),\tau(\alpha))}\,.$$

Notice that, for every contract σ, the set $\{\sigma(\varphi) \mid \sigma \stackrel{\varphi}{\longmapsto}\}$ is finite because σ is regular. Hence, for every $(\sigma,\tau) \in \mathscr{W}$, the set of pairs $\{(\sigma(\varphi),\tau(\varphi)) \mid \sigma \stackrel{\varphi}{\longmapsto}, \tau \stackrel{\varphi}{\longmapsto}\}$ is also finite, hence each $f_{(\sigma,\tau)}$ is well defined, regular, and, by construction, also contractive. Now we prove that

$$\mathscr{S} \stackrel{\text{def}}{=} \{(\sigma, f_{(\sigma,\tau)}(\tau)) \mid (\sigma,\tau) \in \mathscr{W}\}$$

is a strong subcontract relation. Let $(\sigma, f_{(\sigma,\tau)}(\tau)) \in \mathscr{S}$. As regards condition (1) in the definition of coinductive strong subcontract relation, assume $\tau \Downarrow \mathrm{R}$. By definition of $\mathrm{A}(\sigma,\tau)$ there exists $\mathrm{S_R} \subseteq \mathrm{R}$ such that $\sigma \Downarrow \mathrm{S_R}$ and also $\mathrm{S_R} \subseteq \mathrm{A}(\sigma,\tau)$, so we conclude $\mathrm{S_R} \subseteq f_{(\sigma,\tau)}(\mathrm{R})$. As regards condition (2) in the definition of coinductive strong subcontract relation, assume $f_{(\sigma,\tau)}(\tau) \stackrel{\alpha}{\longmapsto}$. Then $\tau \stackrel{\alpha}{\longmapsto}$ and there exists $\mathrm{S_R}$ such that $\sigma \Downarrow \mathrm{S_R}$ and $\alpha \in \mathrm{S_R}$, hence we obtain $\sigma \stackrel{\alpha}{\longmapsto}$ and $\mathrm{A}(\sigma,\tau) \neq \emptyset$. From $(\sigma,\tau) \in \mathscr{W}$ we derive $(\sigma(\alpha),\tau(\alpha)) \in \mathscr{W}$ so we conclude $(\sigma(\alpha), f_{(\sigma(\alpha),\tau(\alpha))}(\tau(\alpha))) \in \mathscr{S}$ by definition of $\mathscr{S}$. □

In terms of compliance this theorem yields the following corollary:

Corollary 3.10. *$\rho \dashv\!\!\!\dashv \sigma$ if and only if there exists a filter f such that $\rho \dashv f(\sigma)$*

Since $\dashv$ ensures that a client will either continuously interact or successfully terminate with a strongly compliant service, this corollary tells us that filters are the operational device that ensures the same property in case of weak compliance. Properties of client/service interactions are formally stated in §4.

3.2.1 *Examples of filters.* 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 synchronization on a is possible, and that if it occurs the client will terminate. The point is then to ensure that this synchronization will indeed occur and that the channel b will not be selected instead, which would lead to a 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.

We have already hinted in the introduction that to prove an inclusion such as $a.b \preceq (a.(a+b))+b.c$ filters must be able to selectively block along the computation, as b must be blocked only at the first step of the interaction and a only at the second step of the interaction. In this case the sought behavior is obtained by the single-threaded filter $f = a.b$ which when applied to the contract on the right yields the one on the left. Such fine-grainedness of filters is useful also in practice. Consider again the last example of §2.2.2, where we extended the service by a "1-click ordering"

Table I. Deduction system for the weak subcontract relation.

(E1)	$\sigma + \sigma = \sigma$	(WEAK)	$\dfrac{f : \sigma \leq \tau \quad g \wedge I(\tau) \leqslant f}{f \vee g : \sigma \leq \tau}$
(E2)	$\sigma + \tau = \tau + \sigma$		
(E3)	$\sigma + (\sigma' + \sigma'') = (\sigma + \sigma') + \sigma''$		
(E4)	$\sigma + \mathbf{0} = \sigma$	(TRANS)	$\dfrac{f : \sigma \leq \sigma' \quad g : \sigma' \leq \sigma''}{f \wedge g : \sigma \leq \sigma''}$
(I1)	$\sigma \oplus \sigma = \sigma$		
(I2)	$\sigma \oplus \tau = \tau \oplus \sigma$		
(I3)	$\sigma \oplus (\sigma' \oplus \sigma'') = (\sigma \oplus \sigma') \oplus \sigma''$	(PREFIX)	$\dfrac{f : \sigma \leq \tau}{\alpha.f : \alpha.\sigma \leq \alpha.\tau}$
(D1)	$\sigma + (\sigma' \oplus \sigma'') = (\sigma + \sigma') \oplus (\sigma + \sigma'')$		
(D2)	$\sigma \oplus (\sigma' + \sigma'') = (\sigma \oplus \sigma') + (\sigma \oplus \sigma'')$	(ICHOICE)	$\dfrac{f : \sigma \leq \sigma' \quad f : \tau \leq \tau'}{f : \sigma \oplus \tau \leq \sigma' \oplus \tau'}$
(D3)	$\alpha.\sigma + \alpha.\tau = \alpha.(\sigma \oplus \tau)$		
(D4)	$\alpha.\sigma \oplus \alpha.\tau = \alpha.(\sigma \oplus \tau)$		
(MUST)	$I(\sigma) : \sigma \oplus \tau \leq \sigma$	(ECHOICE)	$\dfrac{f : \sigma \leq \sigma' \quad f : \tau \leq \tau'}{f : \sigma + \tau \leq \sigma' + \tau'}$
(DEPTH)	$\mathbf{0} : \mathbf{0} \leq \sigma$		

capability. We said that backward compatibility can be obtained by filtering out the newly added `Buy` action. But if we slightly expand the contract σ_2'

$$\ldots \overline{\texttt{Catalog}}.(\sigma_2' + \texttt{Logout} + \texttt{AddToCart}.(\sigma_2' + \texttt{Buy}.\sigma_3) + \texttt{Buy}.(\cdots))$$

we observe that there are two distinct occurrences of the `Buy` action. In order to make a service with contract σ_2' implement the contract σ_2 defined in §2.2.2, one must block the `Buy` action offered right after the $\overline{\texttt{Catalog}}$ action, but allow the old `Buy` action in the continuation of `AddToCart` to pass through. This is performed by the filter obtained from σ_1 by replacing $\vee$ for every sum (either internal or external) occurring in it.

3.3 Deduction system for the weak subcontract relation

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 judgment of the form $f : \sigma \leq \tau$ implies that $\sigma \preceq \tau$, and f is a filter that embeds services with contract τ into the world of σ-compliant clients.

The definition of such a 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 3.11. *The* identity filter *for a contract σ, denoted by $I(\sigma)$, is defined as*

$$I(\sigma) \stackrel{\text{def}}{=} \bigvee_{\sigma \stackrel{\alpha}{\longmapsto} \sigma'} \alpha.I(\sigma')$$

It is easy to see that $I(\sigma)(\sigma) \simeq \sigma$.

Then, 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 §3.4, in order to define the "best" filter that proves $\sigma \preceq \tau$.

DEFINITION 3.12 (FILTER ORDER). *The ordering relation on filters $\leqslant$ is the largest relation such that $f \leqslant g$ and $f \xmapsto{\alpha} f'$ implies $g \xmapsto{\alpha} g'$ and $f' \leqslant g'$. We write $f = g$ for $f \leqslant g$ and $g \leqslant f$.*

In terms of filter semantics we have that $f \leqslant g$ if and only if $[\![f]\!] \subseteq [\![g]\!]$. This set-theoretic interpretation gives us the relation between operators $\vee$ and $\wedge$ and filter ordering: the conjunction of two filters is their greatest lower bound, and their disjunction is their least upper bound.

Table I defines the deduction system for $\preceq$. In the table we use a single axiom $\sigma = \tau$ as a shorthand for two axioms $I(\tau) : \sigma \leq \tau$ and $I(\sigma) : \tau \leq \sigma$. The equalities and rule (MUST) are well known since they fully characterize the strong subcontract relation, which coincides with the must preorder [De Nicola and Hennessy 1984; Hennessy 1988]. Notice that in the rule (MUST) no action needs to be actually filtered out and the filter $I(\sigma) \vee I(\tau)$ would work as well. In fact, this is the only axiom for safely enlarging a contract without the intervention of any filter (which is expected since this axiom characterizes strong compliance, where filters are not needed). Rule (DEPTH) formalizes *depth* extension of contracts, where a contract can be prolonged if no action is made visible from the continuation. Rule (WEAK) 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 (TRANS) 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 (ICHOICE) and (ECHOICE) 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.

3.3.1 *Properties.* First of all notice that the deductions of the system we devised in the previous section may be infinite. However valid deductions are regular and contractive. This is a direct consequence of the regularity and contractivity of both contracts and filters. This is easily seen by observing that every deduction rule on the right hand side of Table I deconstructs in its premises either the filter or the contracts that occur in its consequence. This implies that infinite valid derivations are regular and that on every infinite branch of the derivation there are infinitely many applications of the rule (PREFIX).

The deduction system is sound and complete with respect to $\preceq$ *and* the set of filters, in the sense that it proves all and only the pairs 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 3.9.

THEOREM 3.13 (SOUNDNESS). *If $f : \sigma \leq \tau$, then $\sigma \sqsubseteq f(\tau)$.*

PROOF. Let $\mathscr{S}$ be the least relation such that if $f : \sigma \leq \tau$ is derivable, then $(\sigma, f(\tau)) \in \mathscr{S}$. It is sufficient to prove that $\mathscr{S}$ is a coinductive strong subcontract relation. Suppose $f : \sigma \leq \tau$ is derivable, then $(\sigma, f(\tau)) \in \mathscr{S}$. We have to prove that $f(\tau) \Downarrow \mathrm{R}$ implies $\sigma \Downarrow \mathrm{R}'$ and $\mathrm{R}' \subseteq \mathrm{R}$ and that $f(\tau) \stackrel{\alpha}{\longmapsto}$ implies $\sigma \stackrel{\alpha}{\longmapsto}$ and $(\sigma(\alpha), f(\alpha)(\tau(\alpha))) \in \mathscr{S}$. We do so by induction on the maximum depth of an axiom or of an unnested instance of rule (PREFIX) in the derivation tree of $f : \sigma \leq \tau$ and by cases on the last rule applied. Such depth is always finite because contracts are contractive (hence, any infinite branch of the derivation tree must contain infinitely many instances of rule (PREFIX)). In the following we only show the nontrivial cases.

Assume the last rule was (PREFIX). Then $\sigma \equiv \alpha.\sigma'$, $f \equiv \alpha.f'$, $\tau \equiv \alpha.\tau'$, and $f' : \sigma' \leq \tau'$ is derivable (we use $\equiv$ to denote syntactic equality). Suppose $f(\tau) \Downarrow \mathrm{R}$. Then $\mathrm{R} = \{\alpha\}$ and we notice that $\sigma \Downarrow \{\alpha\}$. We also notice that $\tau \stackrel{\alpha}{\longmapsto}$ and $\sigma \stackrel{\alpha}{\longmapsto}$ and that this is the only possible transition for σ and τ. Furthermore, $\sigma(\alpha) \equiv \sigma'$, $f(\alpha) \equiv f'$, and $\tau(\alpha) \equiv \tau'$, and we conclude because $f' : \sigma' \leq \tau'$ is derivable by hypothesis, hence $(\sigma(\alpha), f(\alpha)(\tau(\alpha))) \in \mathscr{S}$ by definition of $\mathscr{S}$.

Assume the last rule was (MUST). Then $\sigma \equiv \sigma' \oplus \tau$ and $f \equiv I(\tau)$. Suppose $f(\tau) \Downarrow \mathrm{R}$. Then $\tau \Downarrow \mathrm{R}$ and $\sigma \Downarrow \mathrm{R}$. Suppose $f(\tau) \stackrel{\alpha}{\longmapsto}$. We have two subcases: if $\sigma' \stackrel{\alpha}{\longmapsto}$, then $\sigma(\alpha) \equiv \sigma'(\alpha) \oplus \tau(\alpha)$ and we conclude $f(\alpha) : \sigma'(\alpha) \oplus \tau(\alpha) \leq \tau(\alpha)$ by (MUST). If $\sigma' \stackrel{\alpha}{\not\longmapsto}$, then $\sigma(\alpha) \equiv \tau(\alpha)$, hence we conclude by reflexivity of $\leq$ (indeed $\sigma = \sigma \oplus \sigma$ and $I(\sigma) : \sigma \oplus \sigma \leq \sigma$).

Assume the last rule was (DEPTH). Then $\sigma \equiv \mathbf{0}$ and $f \equiv \mathbf{0}$. The condition on ready sets of $f(\tau)$ trivially holds because $\sigma \Downarrow \emptyset$. Furthermore $f(\tau) \stackrel{\alpha}{\not\longmapsto}$ for every α.

Assume the last rule was (WEAK). Then $f \equiv f' \vee g$, $f' : \sigma \leq \tau$, and $g \wedge I(\tau) \leqslant f'$. Suppose $f(\tau) \Downarrow \mathrm{R}$. Since by definition f is less restrictive than f', there is a $\mathrm{R}' \subseteq \mathrm{R}$ such that $f'(\tau) \Downarrow \mathrm{R}'$. By induction hypothesis, σ has a ready set R'' such that $\mathrm{R}'' \subseteq \mathrm{R}'$, hence we conclude $\mathrm{R}'' \subseteq \mathrm{R}$. Suppose $f(\tau) \stackrel{\alpha}{\longmapsto}$. We have two subcases. If $g \stackrel{\alpha}{\longmapsto}$, then $f(\alpha) \equiv f'(\alpha) \vee g(\alpha)$ and $g(\alpha) \wedge I(\tau(\alpha)) \leqslant f'(\alpha)$. By induction hypothesis $\sigma \stackrel{\alpha}{\longmapsto}$ and $f'(\alpha) : \sigma(\alpha) \leq \tau(\alpha)$ is derivable, hence we derive $f(\alpha) : \sigma(\alpha) \leq \tau(\alpha)$ by (WEAK), so we conclude $(\sigma(\alpha), f(\alpha)(\tau(\alpha))) \in \mathscr{S}$ by definition of $\mathscr{S}$. If $g \stackrel{\alpha}{\not\longmapsto}$, then $f(\alpha) \equiv f'(\alpha)$. By induction hypothesis $\sigma \stackrel{\alpha}{\longmapsto}$ and $f'(\alpha) : \sigma(\alpha) \leq \tau(\alpha)$, hence $(\sigma(\alpha), f(\alpha)(\tau(\alpha))) \in \mathscr{S}$ by definition of $\mathscr{S}$.

Assume the last rule was (TRANS). Then $f \equiv f' \wedge g$, $f' : \sigma \leq \sigma'$, $g : \sigma' \leq \tau$. Suppose $\tau \Downarrow \mathrm{R}$. By induction hypothesis σ' has a ready set R' such that $\mathrm{R}' \subseteq g(\mathrm{R})$. By induction hypothesis σ has a ready set R'' such that $\mathrm{R}'' \subseteq f'(\mathrm{R}') \subseteq f'(g(\mathrm{R})) = f(\mathrm{R})$. Suppose $f(\tau) \stackrel{\alpha}{\longmapsto}$. Then $g(\tau) \stackrel{\alpha}{\longmapsto}$. By induction hypothesis $\sigma' \stackrel{\alpha}{\longmapsto}$ and $g(\alpha) : \sigma'(\alpha) \leq \tau(\alpha)$ is derivable. From $f(\tau) \stackrel{\alpha}{\longmapsto}$ we also derive $f' \stackrel{\alpha}{\longmapsto}$, hence $f'(\sigma') \stackrel{\alpha}{\longmapsto}$. Again by induction hypothesis, $\sigma \stackrel{\alpha}{\longmapsto}$ and $f'(\alpha) : \sigma(\alpha) \leq \sigma'(\alpha)$ is derivable. By (TRANS) we conclude that $f(\alpha) : \sigma(\alpha) \leq \tau(\alpha)$ is derivable.

Assume the last rule was (ICHOICE). Then $\sigma \equiv \sigma' \oplus \tau'$, $\tau \equiv \sigma'' \oplus \tau''$, $f : \sigma' \leq \sigma''$,

and $f : \tau' \leq \tau''$. Suppose $f(\tau) \Downarrow \mathrm{R}$ and assume, without loss of generality, that $f(\tau'') \Downarrow \mathrm{R}$. By induction hypothesis we obtain R' such that $\tau' \Downarrow \mathrm{R}'$ and $\mathrm{R}' \subseteq \mathrm{R}$. We conclude by observing that $\sigma \Downarrow \mathrm{R}'$. Suppose $f(\tau) \stackrel{\alpha}{\longmapsto}$. We have three subcases, depending on which contracts between σ'' and τ'' admit α-successors. Assume $\sigma'' \stackrel{\alpha}{\longmapsto}$ and $\tau'' \stackrel{\alpha}{\longmapsto}$. By induction hypothesis $\sigma' \stackrel{\alpha}{\longmapsto}$ and $\tau' \stackrel{\alpha}{\longmapsto}$ and $f(\alpha) : \sigma'(\alpha) \leq \sigma''(\alpha)$ is derivable and $f(\alpha) : \tau'(\alpha) \leq \tau''(\alpha)$ is derivable. Then we conclude $f(\alpha) : \sigma(\alpha) \leq \tau(\alpha)$ is also derivable because $\sigma(\alpha) \equiv \sigma'(\alpha) \oplus \tau'(\alpha)$ and $\tau(\alpha) \equiv \sigma''(\alpha) \oplus \tau''(\alpha)$. On the other hand, suppose $\sigma'' \stackrel{\alpha}{\longmapsto}$ but $\tau'' \stackrel{\alpha}{\not\longmapsto}$. By induction hypothesis $\sigma' \stackrel{\alpha}{\longmapsto}$ and $f(\alpha) : \sigma'(\alpha) \leq \sigma''(\alpha)$ is derivable. We distinguish two further subcases. Either (i) $\tau' \stackrel{\alpha}{\longmapsto}$ or (ii) $\tau' \stackrel{\alpha}{\not\longmapsto}$. In subcase (i) we have $\sigma(\alpha) \equiv \sigma'(\alpha) \oplus \tau'(\alpha)$. By (Must) we derive $I(\sigma'(\alpha)) : \sigma(\alpha) \leq \sigma'(\alpha)$, from $f(\alpha) \wedge I(\sigma'(\alpha)) \leqslant I(\sigma'(\alpha))$ and (weak) we obtain $f(\alpha) : \sigma(\alpha) \leq \sigma'(\alpha)$ and now we conclude $f(\alpha) : \sigma(\alpha) \leq \tau(\alpha)$ by (trans) and noticing that $\tau(\alpha) \equiv \sigma''(\alpha)$. In subcase (ii) we have $\sigma(\alpha) \equiv \sigma'(\alpha)$ and now $f(\alpha) : \sigma(\alpha) \leq \tau(\alpha)$ is derivable by hypothesis.

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

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 rewrite an arbitrary contract to an equivalent one which is in normal form by using the axioms.

As regards the actual definition of the normal form, we 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 $I(\sigma)$ does not change and the new ready sets contain older ones: for example, $\sigma \oplus \tau \simeq \sigma \oplus \tau \oplus (\sigma + \tau)$. 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 3.14 (normal form [Hennessy 1988]). *For any contract σ, we define its saturated set of ready sets:*

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

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

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

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). Notice that $\mathtt{nf}(\sigma)$ is well defined because σ is regular, hence $\{\sigma(\varphi) \mid \sigma \stackrel{\varphi}{\longmapsto}\}$ is finite.

The normal form enjoys the following 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

Table II. Derived rules.

(S1)	$\sigma \oplus \tau = \sigma \oplus \tau \oplus (\sigma + \tau)$	(C-PREFIX)	$\dfrac{\sigma = \sigma'}{\alpha.\sigma = \alpha.\sigma'}$
(S2)	$\sigma \oplus (\sigma + \tau + \rho) = \sigma \oplus (\sigma + \tau) \oplus (\sigma + \tau + \rho)$		
(CO)	$(\alpha.\sigma' + \tau') \oplus (\alpha.\sigma'' + \tau'') = (\alpha.(\sigma' \oplus \sigma'') + \tau') \oplus (\alpha.(\sigma' \oplus \sigma'') + \tau'')$	(C-ECHOICE)	$\dfrac{\sigma = \sigma' \quad \tau = \tau'}{\sigma + \tau = \sigma' + \tau'}$
(WIDTH)	$\dfrac{I(\sigma) \wedge I(\tau) \leqslant \mathbf{0}}{I(\sigma) : \sigma \leq \sigma + \tau}$	(C-ICHOICE)	$\dfrac{\sigma = \sigma' \quad \tau = \tau'}{\sigma \oplus \tau = \sigma' \oplus \tau'}$

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.

To prove that every contract can be rewritten to an equivalent one in normal form it is useful to derive a handful of auxiliary axioms and rules (Table II) that will be fundamental in the following. Axioms (S1) and (S2) will be used for saturating ready sets as required by the definition of normal form. Axiom (CO) shows that it is possible to rewrite a contract so that all the continuations under the same prefix α are equal. Rules (C-PREFIX), (C-ICHOICE), and (C-ECHOICE) are strengthened versions of rules (PREFIX), (ICHOICE), and (ECHOICE) showing the congruence properties of $=$ with respect to the prefix and the two choices. Such rules will allow us to replace equivalent contracts in arbitrary contexts. Finally, rule (WIDTH) states that a service can be extended with additional capabilities, provided that such capabilities are disjoint from those that were available before the extension.

LEMMA 3.15. *The axioms and rules in Table II are derivable from those in Table I.*

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

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

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

As regards (S2):

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

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

As regards (CO):

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

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

Proving (C-PREFIX) is trivial. As regards (C-ECHOICE), observe that from $I(\sigma') : \sigma \leq \sigma'$ and $I(\tau') \wedge I(\sigma') \leqslant I(\sigma')$ we derive $I(\sigma') \vee I(\tau') : \sigma \leq \sigma'$ by an application of (WEAK). Similarly we can derive $I(\sigma') \vee I(\tau') : \tau \leq \tau'$, hence we can apply (ECHOICE) and derive $I(\sigma') \vee I(\tau') : \sigma + \tau \leq \sigma' + \tau'$. By a similar argument we can also derive $I(\sigma) \vee I(\tau) : \sigma' + \tau' \leq \sigma + \tau$, hence $\sigma + \tau = \sigma' + \tau'$. Rule (C-ICHOICE) is analogous.

As regards (WIDTH), from the axiom $\mathbf{0} : \mathbf{0} \leq \tau$ and the hypothesis $I(\sigma) \wedge I(\tau) \leqslant \mathbf{0}$ we derive $I(\sigma) : \mathbf{0} \leq \tau$. From $I(\sigma) : \sigma \leq \sigma$ and applying (ECHOICE) we conclude $I(\sigma) : \sigma + \mathbf{0} \leq \sigma + \tau$, hence $I(\sigma) : \sigma \leq \sigma + \tau$. □

We are now ready to prove that every contract can be rewritten into its own normal form.

LEMMA 3.16 (NORMAL FORM). *The judgment $\sigma = \mathtt{nf}(\sigma)$ is derivable.*

PROOF. We define the *head normal form* of σ as $\mathtt{hnf}(\sigma) \stackrel{\text{def}}{=} \bigoplus_{\mathrm{R} \in \mathscr{R}(\sigma)} \sum_{\alpha \in \mathrm{R}} \alpha.\sigma(\alpha)$. It is sufficient to prove that $\sigma = \mathtt{hnf}(\sigma)$ is derivable because then what remains to prove is $\sigma(\alpha) = \mathtt{hnf}(\sigma(\alpha))$, but since σ is regular the number of these proofs is the same as the cardinality of $\{\sigma(\varphi) \mid \sigma \stackrel{\varphi}{\longmapsto}\}$, which is finite, hence the (possibly infinite) proof of $\sigma = \mathtt{nf}(\sigma)$ is regular.

We prove $\sigma = \mathtt{hnf}(\sigma)$ by induction on the maximum depth of a topmost prefix in σ and by cases on the structure of σ. If $\sigma \equiv \mathbf{0}$, then σ is already in head normal form.

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

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

$$
\begin{aligned}
\sigma &= (\textstyle\bigoplus_{\mathrm{R}_1 \in \mathscr{R}(\sigma_1)} \sum_{\alpha \in \mathrm{R}_1} \alpha.\sigma_1(\alpha)) + (\bigoplus_{\mathrm{R}_2 \in \mathscr{R}(\sigma_2)} \sum_{\beta \in \mathrm{R}_2} \beta.\sigma_2(\beta)) && (1)\\
&= \textstyle\bigoplus_{\mathrm{R}_1 \in \mathscr{R}(\sigma_1), \mathrm{R}_2 \in \mathscr{R}(\sigma_2)} (\sum_{\alpha \in \mathrm{R}_1} \alpha.\sigma_1(\alpha) + \sum_{\beta \in \mathrm{R}_2} \beta.\sigma_2(\beta)) && (2)\\
&= \textstyle\bigoplus_{\mathrm{R}_1 \in \mathscr{R}(\sigma_1), \mathrm{R}_2 \in \mathscr{R}(\sigma_2)} \sum_{\alpha \in \mathrm{R}_1 \cup \mathrm{R}_2} \alpha.\sigma(\alpha) && (3)\\
&= \textstyle\bigoplus_{\mathrm{R} \in \mathscr{R}(\sigma)} \sum_{\alpha \in \mathrm{R}} \alpha.\sigma(\alpha) && (4)
\end{aligned}
$$

where (1) is justified by the induction hypothesis and congruence rules, (2) is justified by the repeated use of (D1), (3) is justified by (CO), and (4) follows from $\mathscr{R}(\sigma) = \{\mathrm{R}_1 \cup \mathrm{R}_2 \mid \mathrm{R}_1 \in \mathscr{R}(\sigma_1), \mathrm{R}_2 \in \mathscr{R}(\sigma_2)\}$. Indeed, if $\mathrm{R} \in \mathscr{R}(\sigma)$, then there exist R'_1 and R'_2 such that $\sigma_1 \Downarrow \mathrm{R}'_1$ and $\sigma_2 \Downarrow \mathrm{R}'_2$ and $\mathrm{R}'_1 \cup \mathrm{R}'_2 \subseteq \mathrm{R}$. Now $\mathrm{R}'_1 \subseteq \mathrm{R} \cap \mathsf{init}(\sigma_1) \subseteq \mathsf{init}(\sigma_1)$ and $\mathrm{R}'_2 \subseteq \mathrm{R} \cap \mathsf{init}(\sigma_2) \subseteq \mathsf{init}(\sigma_2)$, hence

$\mathrm{R} \cap \mathsf{init}(\sigma_1) \in \mathscr{R}(\sigma_1)$ and $\mathrm{R} \cap \mathsf{init}(\sigma_2) \in \mathscr{R}(\sigma_2)$. We conclude by observing that $(\mathrm{R} \cap \mathsf{init}(\sigma_1)) \cup (\mathrm{R} \cap \mathsf{init}(\sigma_2)) = \mathrm{R}$ because $\mathrm{R} \subseteq \mathsf{init}(\sigma_1) \cup \mathsf{init}(\sigma_2)$. On the other hand, let $\mathrm{R}_1 \in \mathscr{R}(\sigma_1)$ and $\mathrm{R}_2 \in \mathscr{R}(\sigma_2)$. Then there exist ready sets R'_1 and R'_2 of respectively σ_1 and σ_2 such that $\mathrm{R}'_1 \subseteq \mathrm{R}_1 \subseteq \mathsf{init}(\sigma_1)$ and $\mathrm{R}'_2 \subseteq \mathrm{R}_2 \subseteq \mathsf{init}(\sigma_2)$. Hence $\mathrm{R}'_1 \cup \mathrm{R}'_2 \subseteq \mathrm{R}_1 \cup \mathrm{R}_2 \subseteq \mathsf{init}(\sigma_1) \cup \mathsf{init}(\sigma_2)$ and we conclude $\mathrm{R}_1 \cup \mathrm{R}_2 \in \mathscr{R}(\sigma)$ by observing that $\sigma \Downarrow \mathrm{R}'_1 \cup \mathrm{R}'_2$ and $\mathsf{init}(\sigma) = \mathsf{init}(\sigma_1) \cup \mathsf{init}(\sigma_2)$.

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

$$\begin{aligned} \sigma &= (\textstyle\bigoplus_{\mathrm{R}_1 \in \mathscr{R}(\sigma_1)} \sum_{\alpha \in \mathrm{R}_1} \alpha.\sigma_1(\alpha)) \oplus (\bigoplus_{\mathrm{R}_2 \in \mathscr{R}(\sigma_2)} \sum_{\beta \in \mathrm{R}_2} \beta.\sigma_2(\beta)) && (1)\\ &= (\textstyle\bigoplus_{\mathrm{R}_1 \in \mathscr{R}(\sigma_1)} \sum_{\alpha \in \mathrm{R}_1} \alpha.\sigma(\alpha)) \oplus (\bigoplus_{\mathrm{R}_2 \in \mathscr{R}(\sigma_2)} \sum_{\beta \in \mathrm{R}_2} \beta.\sigma(\beta)) && (2)\\ &= \textstyle\bigoplus_{\mathrm{R} \in \mathscr{R}(\sigma)} \sum_{\alpha \in \mathrm{R}} \alpha.\sigma(\alpha) && (3) \end{aligned}$$

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

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

THEOREM 3.17 (COMPLETENESS). *If $\sigma \sqsubseteq f(\tau)$, then $f : \sigma \leq \tau$.*

PROOF. By Lemma 3.16 we can assume that σ and τ are in normal form. Additionally, for the sake of simplicity we identify $\sigma(\alpha)$, $\tau(\alpha)$, and $f(\tau)$ with their corresponding normal forms (indeed $\mathtt{nf}(\sigma(\alpha))$, $\mathtt{nf}(\tau(\alpha))$, and $\mathtt{nf}(f(\tau(\alpha)))$ can be obtained from $\sigma(\alpha)$, $\tau(\alpha)$, and $f(\tau)$ by repeated use of (I1)).

Let $P(f, \sigma, \tau)$ stand for the proof tree whose conclusion is $f : \sigma \leq \tau$. Below we will show that $P(f, \sigma, \tau)$ can be built provided that proof trees $P(f(\alpha), \sigma(\alpha), \tau(\alpha))$ are available for all α such that $\sigma \xrightarrow{\alpha}$ and $f(\tau) \xrightarrow{\alpha}$. However, the set $\{(f(\varphi), \sigma(\varphi), \tau(\varphi)) \mid f \xrightarrow{\varphi}, \sigma \xrightarrow{\varphi}, \tau \xrightarrow{\varphi}\}$ is finite because f, σ, and τ are regular. Hence, we overall need to show how to build a finite number of proofs $P(f(\varphi), \sigma(\varphi), \tau(\varphi))$ and so the possibly infinite proof of $P(f, \sigma, \tau)$ is also regular.

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

For the remaining cases, assume

$$\sigma \equiv \textstyle\bigoplus_{\mathrm{R} \in \mathscr{R}(\sigma)} \sum_{\alpha \in \mathrm{R}} \alpha.\sigma(\alpha) \qquad \text{and} \qquad \tau \equiv \bigoplus_{\mathrm{S} \in \mathscr{R}(\tau)} \sum_{\alpha \in \mathrm{S}} \alpha.\tau(\alpha)$$

and assume $f(\tau) \xrightarrow{\alpha}$. From $\sigma \sqsubseteq f(\tau)$ we have $\sigma \xrightarrow{\alpha}$ and from the proof $P(f(\alpha), \sigma(\alpha), \tau(\alpha))$ we derive

$$f(\alpha) : \sigma(\alpha) \leq \tau(\alpha)$$

then, by (PREFIX),

$$\alpha.f(\alpha) : \alpha.\sigma(\alpha) \leq \alpha.\tau(\alpha)\,.$$

Now assume $\tau \Downarrow \mathrm{R}$. From $\sigma \sqsubseteq f(\tau)$ and the fact that σ and τ are in head normal form we have $\sigma \Downarrow f(\mathrm{R})$. Let $f_{\mathrm{R}} \stackrel{\text{def}}{=} \bigvee_{\alpha \in f(\mathrm{R})} \alpha.f(\alpha)$ and notice that $f_{\mathrm{R}} \wedge \alpha.I(\tau(\alpha)) \leqslant \alpha.f(\alpha)$. Hence, by (WEAK),

$$f_{\mathrm{R}} : \alpha.\sigma(\alpha) \leq \alpha.\tau(\alpha)$$

and, by (ECHOICE),

$$f_{\mathrm{R}} : \textstyle\sum_{\alpha \in f(\mathrm{R})} \alpha.\sigma(\alpha) \leq \sum_{\alpha \in f(\mathrm{R})} \alpha.\tau(\alpha)\,.$$

From $f(\mathrm{R}) \subseteq \mathrm{R}$ and by applying (WIDTH),

$$f_{\mathrm{R}} : \textstyle\sum_{\alpha \in f(\mathrm{R})} \alpha.\sigma(\alpha) \leq \sum_{\alpha \in \mathrm{R}} \alpha.\tau(\alpha)\,.$$

Let $f' \stackrel{\text{def}}{=} \bigvee_{\tau \Downarrow \mathrm{R}'} f_{\mathrm{R}'}$. From $\bigcup_{\tau \Downarrow \mathrm{R}'} f(\mathrm{R}') \cap \mathrm{R} = f(\bigcup_{\tau \Downarrow \mathrm{R}'} \mathrm{R}') \cap \mathrm{R} \subseteq f(\mathrm{R})$ we observe that $f' \wedge \bigvee_{\alpha \in \mathrm{R}} \alpha.I(\tau(\alpha)) \leqslant f_{\mathrm{R}}$. Hence, by (WEAK), by iterating over all the ready sets of τ, and by (ICHOICE), we obtain

$$f' : \textstyle\bigoplus_{\tau \Downarrow \mathrm{R}} \sum_{\alpha \in f(\mathrm{R})} \alpha.\sigma(\alpha) \leq \tau\,.$$

Now

$$f' : \sigma \leq \textstyle\bigoplus_{\tau \Downarrow \mathrm{R}} \sum_{\alpha \in f(\mathrm{R})} \alpha.\sigma(\alpha)$$

by possibly applying (MUST) for removing all the ready sets of σ that have disappeared in $f(\tau)$ hence, by (TRANS), we conclude $f' : \sigma \leq \tau$. In order to prove $f : \sigma \leq \tau$ it is sufficient to apply (WEAK). This is possible because $f \wedge I(\tau) \leqslant f'$. Indeed, assume $f(\tau) \stackrel{\alpha}{\longmapsto}$. Then $\alpha \in \mathrm{R}$ for some $\tau \Downarrow \mathrm{R}$, hence $\sigma \Downarrow f(\mathrm{R})$ and now $\alpha \in f(\mathrm{R})$. So, it must be $f_{\mathrm{R}} \stackrel{\alpha}{\longmapsto}$ from which we conclude $f' \stackrel{\alpha}{\longmapsto}$. □

3.4 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, finding a decision algorithm for a containment relation corresponds to a cut-elimination process (the cut here being the (TRANS) rule in Table I), 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 3.12). 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 the fact of allowing any action besides a and b 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 filters for a given relation, the "best relevant" filter that proves it.

3.4.1 *Filter relevance.* In order to determine the property of "relevance" we have to better understand the role played by the identity filters we introduced in Definition 3.11. It may be noted that the identity filter of a given contract is exactly the tree (prefix-closed set of traces) 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:

$$I(\sigma \oplus \tau) = I(\sigma + \tau) = I(\sigma) \vee I(\tau) \tag{3}$$

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 semantics of the disjunction operator.

The tree of an identity filter accurately represents the idea we mentioned in the Introduction (§1) of a contract's "world": the sets of actions the contract knows of at each step of an interaction. Filter application can be seen as a *projection* of the contract onto the "world" represented by the filter. In the case of a relation $f : \sigma \leq \tau$, f is used to restrict the "world" of τ: then the intuition is that in order to be relevant, f should be defined (only) on that world, which is represented by $I(\tau)$. Indeed, applying to τ the filter f or the filter $f \wedge I(\tau)$ gives 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 $I(\tau)$ according to $\leqslant$.

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

$$\text{if } \sigma \sqsubseteq f(\tau) \text{ and } f \leqslant I(\tau) \text{ then } f \leqslant I(\sigma) \tag{4}$$

Thus relevant filters that prove a relation have to be smaller than the identity filters of *both* contracts. This corresponds to the intuition that f embeds τ services into the "world" of σ: it projects them on something that is included in that world.

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:

$$\text{if } \sigma \sqsubseteq f(\tau) \text{ and } \sigma \sqsubseteq g(\tau) \text{ then } \sigma \sqsubseteq (f \vee g)(\tau) \tag{5}$$

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 $I(\tau)$ 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.

3.4.2 *Algorithm.* The last step is to define an algorithm for building the canonical filter of a relation. In this respect we have to solve two technical problems: the

Table III. Algorithmic deduction system for the weak subcontract relation.

$$\text{(A1)}\quad \frac{(\sigma,\tau)\in\Gamma}{\Gamma\vdash\emptyset:\sigma\preccurlyeq\tau}$$

$$\text{(A2)}\quad \frac{\begin{array}{c}\forall \mathrm{S}\in\mathscr{R}(\tau):\mathrm{S}\cap\mathrm{A}\in\mathscr{R}(\sigma)\\ \mathrm{A}=\{\alpha\in\mathsf{init}(\sigma)\cap\mathsf{init}(\tau)\mid\exists F_\alpha:\Gamma\cup\{(\sigma,\tau)\}\vdash F_\alpha:\sigma(\alpha)\preccurlyeq\tau(\alpha)\}\end{array}}{\Gamma\vdash\bigcup_{\alpha\in\mathrm{A}}F_\alpha\cup\{(\sigma,\tau)\mapsto\mathrm{A}\}:\sigma\preccurlyeq\tau}$$

first problem is due to the fact that, although the algorithm works on infinite terms, it must always be able to report success or failure in finite time. Since contracts are assumed to be regular, we use the well-known technique of memoization for recording pairs of contracts that we deem related, so that they are not processed if found again. To this end we equip judgments with a context Γ storing pairs of contracts already examined. The second problem regards the computation of the canonical filter that proves a relation: it is clear that, in general, such a filter will not be finite and nonetheless we currently have no finite representation for filters. However, a filter is nothing but the specification, at any given time during an interaction, of a (finite) set of actions that are not shielded, namely the set A in Definition 3.1. In particular, because of the regularity of the contracts being related, the set of such A's is necessarily finite. From this set of A's it is trivial to produce a possibly infinite, regular filter. We use F to range over (finite) maps from pairs of contracts to finite sets of actions. For any given contracts σ and τ such that $\sigma\preceq\tau$, the algorithm infers a map F, defined on every pair (σ',τ') reachable from (σ,τ) after some sequence of interactions, which associates with such a pair the finite set of actions A that are not shielded at that point.

Definition 3.18. *We define the relation $\Gamma\vdash F:\sigma\preccurlyeq\tau$ by the inference rules in Table III. We write $F:\sigma\preccurlyeq\tau$ for $\emptyset\vdash F:\sigma\preccurlyeq\tau$.*

Rule (A1) applies when the contracts being processed have already been encountered and assumed to be related. In this case the inferred map F is empty, because the set A of actions that need not be shielded for relating σ and τ is already computed by the instance of the rule where σ and τ occurred for the first time. In rule (A2), the set A represents the largest set of actions leading to continuations which are in the relation, namely A is the largest set of (relevant) actions that need not be shielded for two contracts to be related. The condition on the first line requires A to be large enough, so that when τ is restricted to the actions in A, τ manifests a behavior that is (more deterministic than) that of σ.

It is trivial to see that if F is synthesized by the algorithm and $\{(\sigma',\tau')\mapsto\mathrm{A},(\sigma',\tau')\mapsto\mathrm{A}'\}\subseteq F$, then we have $\mathrm{A}=\mathrm{A}'$. Hence F is indeed a map and from now on we will write $F(\sigma,\tau)$ for the set A associated with (σ,τ) in F. Additionally, $F:\sigma\preccurlyeq\tau$ implies the following properties

(1) $(\sigma,\tau)\in\mathsf{dom}(F)$;

(2) $(\sigma',\tau')\in\mathsf{dom}(F)$ and $\alpha\in F(\sigma',\tau')$ implies $(\sigma'(\alpha),\tau'(\alpha))\in\mathsf{dom}(F)$.

Overall if $F : \sigma \preccurlyeq \tau$, then the map F represents the (possibly infinite) filter $F[\sigma, \tau]$ defined by the equation

$$F[\sigma, \tau] = \bigvee_{\alpha \in F(\sigma,\tau)} \alpha.F[\sigma(\alpha), \tau(\alpha)]\,.$$

The regularity of such filter is a direct consequence of the regularity of σ and τ while contractivity stems from the finiteness of the $\mathsf{init}(\sigma)$ and $\mathsf{init}(\tau)$ sets (thus of the α's) and from the construction of $F[\sigma, \tau]$.

REMARK 3.19. *The rule* (A2) *in Table III is more an algorithmic specification than a deduction rule, insofar as it describes how to compute the set* A*, rather than how to build a proof tree. The following rule*

$$\text{(A2)}\qquad \frac{\forall \alpha \in \mathrm{A} \quad \Gamma \cup \{(\sigma,\tau)\} \vdash F_\alpha : \sigma(\alpha) \preccurlyeq \tau(\alpha)}{\Gamma \vdash \bigcup_{\alpha\in\mathrm{A}} F_\alpha \cup \{(\sigma,\tau) \mapsto \mathrm{A}\} : \sigma \preccurlyeq \tau} \quad \begin{array}{c} \forall \beta \in (\mathsf{init}(\sigma) \cap \mathsf{init}(\tau)) \setminus \mathrm{A} \\ \Gamma \cup \{(\sigma,\tau)\} \nvdash F : \sigma(\beta) \preccurlyeq \tau(\beta) \\ \forall \mathrm{S} \in \mathscr{R}(\tau) : \mathrm{S} \cap \mathrm{A} \in \mathscr{R}(\sigma) \end{array}$$

is its proof theoretic counterpart.

3.4.3 *Properties.* The algorithm described in Definition 3.18 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 3.20 (FILTER RELEVANCE). *If $F : \sigma \preccurlyeq \tau$, then $F[\sigma, \tau] \leqslant I(\tau)$.*

PROOF. By definition of $F[\sigma, \tau]$ it is trivial to verify that $F[\sigma, \tau] \overset{\varphi}{\longmapsto}$ implies $\tau \overset{\varphi}{\longmapsto}$, from which relevance follows immediately. □

Before proving soundness, we need an auxiliary (cut-elimination) result stating that an hypothesis $\sigma \preceq \tau$ that is necessary for proving $\sigma' \preceq \tau'$ can be discharged if $\sigma \preceq \tau$ is itself provable.

PROPOSITION 3.21. *If (1) $\Gamma \vdash F : \sigma \preccurlyeq \tau$ and (2) $\Gamma \cup \{(\sigma,\tau)\} \vdash F' : \sigma' \preccurlyeq \tau'$, then there exists F'' such that $F' \subseteq F''$ and $\Gamma \vdash F'' : \sigma' \preccurlyeq \tau'$.*

PROOF. We reason by induction on the derivation tree of (2) and by cases on the last rule applied. Assume the last rule was (A1). Then $(\sigma', \tau') \in \Gamma \cup \{(\sigma,\tau)\}$ and $F' = \emptyset$. We distinguish two subcases: if $(\sigma', \tau') \in \Gamma$, then we conclude immediately by (A1) and by taking $F'' = \emptyset$; if $\sigma' \equiv \sigma$ and $\tau' \equiv \tau$, then we conclude by hypothesis (1) and by taking $F'' = F$. Assume the last rule was (A2) and let A be the set determined in the premises of the rule. For every $\alpha \in \mathrm{A}$ we have that there exists F_α such that $\Gamma' \cup \{(\sigma,\tau)\} \vdash F_\alpha : \sigma'(\alpha) \preccurlyeq \tau'(\alpha)$ where $\Gamma' = \Gamma \cup \{(\sigma', \tau')\}$ and $F' = \bigcup_{\alpha\in\mathrm{A}} F_\alpha \cup \{(\sigma', \tau') \mapsto \mathrm{A}\}$. From (1) and $\Gamma \subseteq \Gamma'$ we derive $\Gamma' \vdash F : \sigma \preccurlyeq \tau$. By induction hypothesis there exists F'_α such that $F_\alpha \subseteq F'_\alpha$ and $\Gamma' \vdash F'_\alpha : \sigma'(\alpha) \preccurlyeq \tau'(\alpha)$ for every $\alpha \in \mathrm{A}$. Hence we can apply rule (A2) and conclude that $\Gamma \vdash F'' : \sigma' \preccurlyeq \tau'$ where $F'' = \bigcup_{\alpha\in\mathrm{A}} F'_\alpha \cup \{(\sigma', \tau') \mapsto \mathrm{A}\}$, observing that $F' \subseteq F''$. □

THEOREM 3.22 (SOUNDNESS). *If $F : \sigma \preccurlyeq \tau$ then $\sigma \sqsubseteq F[\sigma, \tau](\tau)$.*

PROOF. We prove that $\mathscr{S} \overset{\text{def}}{=} \{(\sigma, F[\sigma, \tau](\tau)) \mid F : \sigma \preccurlyeq \tau\}$ is a coinductive strong subcontract relation. Let $(\sigma, \tau') \in \mathscr{S}$, then there exist F and τ such that $F : \sigma \preccurlyeq \tau$ and $\tau' \equiv F[\sigma, \tau](\tau)$. As regards condition 1 in the definition of coinductive strong

subcontract relation, let $\tau \Downarrow \mathrm{S}$. Then $F[\sigma,\tau](\tau) \Downarrow \mathrm{S} \cap F(\sigma,\tau)$ and from $\mathrm{S} \cap F(\sigma,\tau) \in \mathscr{R}(\sigma)$ we conclude that there exists R such that $\sigma \Downarrow \mathrm{R}$ and $\mathrm{R} \subseteq \mathrm{S} \cap F(\sigma,\tau)$. As regards condition 2, assume $F[\sigma,\tau](\tau) \overset{\alpha}{\longmapsto}$. By definition of $F[\sigma,\tau]$ we also have $\sigma \overset{\alpha}{\longmapsto}$ and, for every $\alpha \in F(\sigma,\tau)$, there exists F_α such that $\{(\sigma,\tau)\} \vdash F_\alpha : \sigma(\alpha) \preccurlyeq \tau(\alpha)$. By Proposition 3.21 there exists F'_α such that $F_\alpha \subseteq F'_\alpha$ and $F'_\alpha : \sigma(\alpha) \preccurlyeq \tau(\alpha)$. Notice also that $F'_\alpha \subseteq F$ since the former proves $\sigma(\alpha) \preccurlyeq \tau(\alpha)$, the latter $\sigma \preccurlyeq \tau$, and by the uniqueness of the filters derived by the algorithm F'_α must correspond to the α-continuation of F. Hence we conclude $(\sigma(\alpha), \tau'(\alpha)) \in \mathscr{S}$ by observing that $\tau'(\alpha) \equiv F[\sigma,\tau](\tau)(\alpha) \equiv F'_\alpha[\sigma(\alpha), \tau(\alpha)](\tau(\alpha))$ and by definition of $\mathscr{S}$. □

Theorem 3.23 (completeness). *If $\sigma \sqsubseteq g(\tau)$, then there exists F such that $F : \sigma \preccurlyeq \tau$, and $F[\sigma,\tau] \geqslant g \wedge I(\tau)$.*

Proof. First note that if $\sigma \sqsubseteq g(\tau)$, then also $\sigma \sqsubseteq (g \wedge I(\tau))(\tau)$ (applying the conjunction of two filters is like applying one then the other, it projects on the part of the tree common to both), thus we can assume $g \leqslant I(\tau)$ without loss of generality.

The theorem is stated for a memoization environment $\Gamma = \emptyset$ (recall that $F : \sigma \preccurlyeq \tau$ stands for $\emptyset \vdash F : \sigma \preccurlyeq \tau$ and that a memoization environment is a finite set of pairs of contracts). To integrate memoization environments in our proof we generalize the statement and prove that if $\sigma \sqsubseteq g(\tau)$ and $g \leqslant I_\tau$, then for all Γ:

(1) there exists F such that $\Gamma \vdash F : \sigma \preccurlyeq \tau$;

(2) $(\sigma,\tau) \notin \Gamma \Rightarrow \mathsf{init}(g) \subseteq F(\sigma,\tau)$;

Completeness then follows immediately from (1) by taking $\Gamma = \emptyset$, and maximality of the inferred filter is easily deduced from (2).

Let $R(\Gamma,\sigma,\tau) \overset{\text{def}}{=} \{(\sigma(\varphi), \tau(\varphi)) \mid \sigma \overset{\varphi}{\longmapsto}, \tau \overset{\varphi}{\longmapsto}\} \setminus \Gamma$, and note that by regularity of σ and τ, the set is finite. We can thus reason by induction on $R(\Gamma,\sigma,\tau)$ to show the more general property.

Assume $(\sigma,\tau) \in \Gamma$. Note that if $R(\Gamma,\sigma,\tau) = \emptyset$ this is the only possible case. Then we conclude immediately by rule (A1) and by taking $F = \emptyset$. Assume $(\sigma,\tau) \notin \Gamma$ and let $\Gamma' \overset{\text{def}}{=} \Gamma \cup \{(\sigma,\tau)\}$. Suppose $g(\tau) \overset{\alpha}{\longmapsto}$. From the hypothesis $\sigma \sqsubseteq g(\tau)$ we derive $\sigma \overset{\alpha}{\longmapsto}$ and $\sigma(\alpha) \sqsubseteq g(\alpha)(\tau(\alpha))$. We have $R(\Gamma', \sigma(\alpha), \tau(\alpha)) \subsetneq R(\Gamma,\sigma,\tau)$ because (σ,τ) is in the latter and not in the former, hence by induction hypothesis there exists F_α such that $\Gamma' \vdash F_\alpha : \sigma(\alpha) \preccurlyeq \tau(\alpha)$. We can do this for any α such that $g(\tau) \overset{\alpha}{\longmapsto}$, thus because of the hypothesis that $g \leqslant I_\tau$ we have that the set A in the premises of rule (A2) is such that $\mathsf{init}(g) \subseteq \mathrm{A}$. Therefore the F appearing in the conclusion of that rule satisfies property (2) and we just have to check that the condition on the first line of the premises is satisfied. Let $\mathrm{S} \in \mathscr{R}(\tau)$. It contains a ready set S' of τ. Since $\sigma \sqsubseteq g(\tau)$, there exists $\mathrm{R} \subseteq \mathrm{S}' \cap \mathsf{init}(g)$ such that $\sigma \Downarrow \mathrm{R}$. As $\mathsf{init}(g)$ is included in A and S' in S, we also have $\mathrm{R} \subseteq \mathrm{S} \cap \mathrm{A}$. Then $\mathrm{S} \cap \mathrm{A} \in \mathscr{R}(\sigma)$, because A is included in $\mathsf{init}(\sigma)$ by definition. □

Corollary 3.24. *If σ and τ are two contracts, there exists at most one F such that $F : \sigma \preccurlyeq \tau$. Furthermore, if $F : \sigma \preccurlyeq \tau$, then*

$$F[\sigma,\tau] = \max\{g \leqslant I(\tau) \mid \sigma \sqsubseteq g(\tau)\} = \max\{g \leqslant I(\tau) \mid g : \sigma \leq \tau\}\,.$$

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 I is given by the rule (TRANS). This rule intersects filters, that is it minimizes 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 (WEAK) rule, which instead maximizes proofs: therefore we have to set an upper bound to filter maximization, which is embodied by the definition of relevance (therefore it would be more precise to speak of a cut-weakening-elimination process).

PROPOSITION 3.25 (DECIDABILITY). *Given two contracts σ and τ, we can decide whether there exists F such that $F : \sigma \preccurlyeq \tau$.*

PROOF. Trivial consequence of the regularity of σ and τ. □

4. PROCESSES

In this section we relate contracts (which are behavioral types) with processes that implement clients and services. We do not consider any particular process language, nor do we require clients and services to be implemented using the same language. We just require that the observable behavior of processes is described by a labeled transition system and abstracted by a static type system. More precisely we assume that a process language is equipped with a labeled 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 synchronize with the corresponding co-action in the process P is interacting with, or it can be the special action $\mathtt{e}$, by which the client process P signals that it can successfully terminate, 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'. Also, we say that P *diverges* if there exists an infinite sequence $P_0, P_1, \ldots$, such that $P = P_0 \xrightarrow{\boldsymbol{\tau}} P_1 \xrightarrow{\boldsymbol{\tau}} \cdots$.

DEFINITION 4.1 (STRONG PROCESS COMPLIANCE). *Let $P \parallel Q \longrightarrow P' \parallel Q'$ be the least relation defined by the rules:*

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

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 $P \parallel Q \longrightarrow$. A computation of $P \parallel Q$ is a sequence $P \parallel Q = P_0 \parallel Q_0 \longrightarrow P_1 \parallel Q_1 \longrightarrow \cdots$. A computation of $P \parallel Q$ is maximal if either it is infinite or there exists $P_n \parallel Q_n$ such that $P \parallel Q \Longrightarrow P_n \parallel Q_n \not\longrightarrow$.

The client P is strongly compliant *with the service Q, written $P \dashv Q$, if for every configuration $P_i \parallel Q_i$ of every maximal computation there exists $j \geq i$ such that either $P_j \xrightarrow{\alpha} P_{j+1}$ for some α or $P_j \xrightarrow{\boldsymbol{\tau}} \not\longrightarrow$ and $P_j \xrightarrow{\mathtt{e}}$.*

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 is such that either P and Q interact infinitely often, or the client invariably reaches a state in which it is 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 judgment

$$\vdash P : \sigma$$

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 behavior of processes and 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 2.9, the relation $\sqsubseteq$ can be defined without any notion of "successful action" $\mathtt{e}$).

Definition 4.2. *The type system is* consistent *if and only if, for every process P and contract σ, if $\vdash P : \sigma$, then all the following properties hold:*

(1) $P \xrightarrow{\tau} P'$ *implies* $\vdash P' : \sigma'$ *and* $\sigma \sqsubseteq \sigma'$;

(2) $P \xrightarrow{\alpha} P'$ *implies* $\vdash P' : \sigma'$, $\sigma \xrightarrow{\alpha}$, *and* $\sigma(\alpha) \sqsubseteq \sigma'$;

(3) P *diverges implies* $\sigma \Downarrow \emptyset$;

(4) $P \xrightarrow{\tau}\!\!\!\!\!\!/\;\;$ *implies* $\sigma \Downarrow \mathrm{R}$ *and* $\mathrm{R} \subseteq \{\alpha \mid P \xrightarrow{\alpha}\}$.

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 provide that action and the contract of the resulting process P' be (more deterministic than) the contract $\sigma(\alpha)$, which represents all the possible behaviors of P after α. Condition (3) states that a divergent process may be observationally invisible, namely its contract must provide an empty ready set (the process may never be "ready" to perform any action). Finally, condition (4) states that the contract of a stable process should have at least one ready set that provides no more capabilities than those of the process.

The following lemma states that it is possible to replace a client contract ρ 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 4.3. *If $\rho \dashv \sigma$ and $\rho \sqsubseteq \rho'$ then $\rho' \dashv \sigma$.*

Proof. Let $\mathscr{C}$ be a compliance relation such that $(\rho, \sigma) \in \mathscr{C}$ and let $\mathscr{S}$ be a strong subcontract relation such that $(\rho, \rho') \in \mathscr{S}$. It is sufficient to prove that

$$\mathscr{C}' \stackrel{\text{def}}{=} \{(\rho', \sigma) \mid \exists \rho : (\rho, \sigma) \in \mathscr{C} \wedge (\rho, \rho') \in \mathscr{S}\}$$

is a strong compliance relation. Assume $(\rho', \sigma) \in \mathscr{C}'$. Then there exists ρ such that $(\rho, \sigma) \in \mathscr{C}$ and $(\rho, \rho') \in \mathscr{S}$. As regards condition (1) in Definition 2.6, assume $\rho' \Downarrow \mathrm{R}$ and $\sigma \Downarrow \mathrm{S}$. If $\mathtt{e} \in \mathrm{R}$, then the condition is satisfied. Assume $\mathtt{e} \notin \mathrm{R}$. From $(\rho, \rho') \in \mathscr{S}$ there exists $\mathrm{R}' \subseteq \mathrm{R}$ such that $\rho \Downarrow \mathrm{R}'$. In particular, $\mathtt{e} \notin \mathrm{R}'$. From $(\rho, \sigma) \in \mathscr{C}$ we have $\mathtt{co}(\mathrm{R}') \cap \mathrm{S} \neq \emptyset$, hence $\mathtt{co}(\mathrm{R}) \cap \mathrm{S} \neq \emptyset$.

As regards condition (2) in Definition 2.6, assume $\rho' \stackrel{\overline{\alpha}}{\longmapsto}$ and $\sigma \stackrel{\alpha}{\longmapsto}$. From $(\rho, \rho') \in \mathscr{S}$ we derive $\rho \stackrel{\overline{\alpha}}{\longmapsto}$ and $(\rho(\alpha), \rho'(\alpha)) \in \mathscr{S}$. From $(\rho, \sigma) \in \mathscr{C}$ we derive $(\rho(\alpha), \sigma(\alpha)) \in \mathscr{C}$. Hence we conclude $(\rho'(\alpha), \sigma(\alpha)) \in \mathscr{C}'$ by definition of $\mathscr{C}'$. □

Given a consistent type system, the following result states that, given a pair of processes $P \parallel Q$ whose respective contracts comply, and given any two residual processes $P' \parallel Q'$ resulting from $P \parallel Q$, the respective contracts of P' and Q' comply as well.

LEMMA 4.4 (SUBJECT REDUCTION). *If* $\vdash P : \rho$ *and* $\vdash Q : \sigma$ *and* $\rho \dashv \sigma$ *and* $P \parallel Q \longrightarrow P' \parallel Q'$, *then* $\vdash P' : \rho'$ *and* $\vdash Q' : \sigma'$ *and* $\rho' \dashv \sigma'$.

PROOF. We need to consider all the possibilities by which $P \parallel Q$ reduces to $P' \parallel Q'$, namely $P \parallel Q \longrightarrow P' \parallel Q'$. If $P \stackrel{\tau}{\longrightarrow} P'$, then from consistency condition (1) we have $\vdash P' : \rho'$ and $\rho \sqsubseteq \rho'$ and by Lemma 4.3 we conclude $\rho' \dashv \sigma$. If $Q \stackrel{\tau}{\longrightarrow} Q'$, then from consistency condition (1) we have $\vdash Q' : \sigma'$ and $\sigma \sqsubseteq \sigma'$ and by definition of $\sqsubseteq$ we conclude $\rho \dashv \sigma'$. Finally, if $P \stackrel{\overline{\alpha}}{\longrightarrow} P'$ and $Q \stackrel{\alpha}{\longrightarrow} Q'$, then from consistency condition (2) we have that $\vdash P' : \rho'$ and $\vdash Q' : \sigma'$ and $\rho(\alpha) \sqsubseteq \rho'$ and $\sigma(\alpha) \sqsubseteq \sigma'$. By Lemma 4.3 and by definition of $\sqsubseteq$ we conclude $\rho' \dashv \sigma'$. □

The soundness of a consistent type system is ensured by the following result, stating that if the contracts of two processes comply, the corresponding processes comply as well, guaranteeing that either the two processes synchronize infinitely many times or the client successfully terminates.

THEOREM 4.5. *If* $\vdash P : \rho$ *and* $\vdash Q : \sigma$ *and* $\rho \dashv \sigma$ *then* $P \dashv Q$.

PROOF. Because of Lemma 4.4 we only need to consider the cases when $P \parallel Q \not\longrightarrow$ or $P \not\longrightarrow$ and Q diverges. Indeed, from $\rho \dashv \sigma$ we derive that $\rho \Downarrow \mathrm{R}$ implies $\mathrm{R} \neq \emptyset$, hence P cannot diverge, for otherwise we would have $\rho \Downarrow \emptyset$ by consistency condition (3). Let $P \parallel Q \not\longrightarrow$ and assume, by contradiction, that $P \stackrel{\mathsf{e}}{\not\longrightarrow}$. From $P \parallel Q \not\longrightarrow$ we have that whenever $P \stackrel{\alpha}{\longrightarrow}$ we have $Q \stackrel{\overline{\alpha}}{\not\longrightarrow}$. From consistency condition (4) there exist R and S such that $\rho \Downarrow \mathrm{R}$ and $\sigma \Downarrow \mathrm{S}$ and $\mathtt{co}(\mathrm{R}) \cap \mathrm{S} = \emptyset$ and $\mathsf{e} \notin \mathrm{R}$, but this is absurd from the hypothesis that $\rho \dashv \sigma$. Hence $P \stackrel{\mathsf{e}}{\longrightarrow}$. Assume that $P \not\longrightarrow$ and Q diverges. By consistency condition (3) we derive $\sigma \Downarrow \emptyset$, hence $\rho \Downarrow \mathrm{R}$ implies $\mathsf{e} \in \mathrm{R}$. From consistency condition (4) we conclude $P \stackrel{\mathsf{e}}{\longrightarrow}$. □

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 ability to evolve autonomously. The labeled transition system

of the language is consequently enriched with the following two inference rules:

$$\frac{P \xrightarrow{\alpha} P' \qquad f \overset{\alpha}{\longmapsto} f'}{f[P] \xrightarrow{\alpha} f'[P']}\ (\textsc{Filter1}) \qquad \frac{P \xrightarrow{\tau} P'}{f[P] \xrightarrow{\tau} f[P']}\ (\textsc{Filter2})$$

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

$$\frac{\vdash P : \sigma}{\vdash f[P] : f(\sigma)}\ (\textsc{TypeFilter})$$

preserves the consistency of the type system.

Proposition 4.6. *A consistent type system enriched with rule* (TypeFilter) *results in another consistent type system.*

Proof. Let $\vdash P : \sigma$. As regards consistency condition (1), assume $P \xrightarrow{\tau} P'$ and $\vdash P : \sigma'$. Then $\sigma \sqsubseteq \sigma'$ implies $f(\sigma) \sqsubseteq f(\sigma')$ by Proposition 3.8. As regards consistency condition (2), assume that $P \xrightarrow{\alpha} P'$ and $\vdash P' : \sigma'$. There are two possibilities: if $f \overset{\alpha}{\not\longmapsto}$, then $f[P] \xrightarrow{\alpha}$ and there is nothing to prove. If $f \overset{\alpha}{\longmapsto} f'$, then $\sigma(\alpha) \sqsubseteq \sigma'$. Now we conclude $f(\sigma)(\alpha) = f'(\sigma(\alpha)) \sqsubseteq f'(\sigma')$. As regards consistency condition (3), assume that $f[P]$ diverges. Then P diverges and we must have $\sigma \Downarrow \emptyset$. We immediately conclude $f(\sigma) \Downarrow \emptyset$. Finally, as regards consistency condition (4), assume that $f[P] \xrightarrow{\tau}$. Then $P \xrightarrow{\tau}$. We derive $\sigma \Downarrow \mathrm{R}$ where $\mathrm{R} \subseteq \{\alpha \mid P \xrightarrow{\alpha}\}$, hence $f(\sigma) \Downarrow f(\mathrm{R})$ and we conclude by observing that $f(\mathrm{R}) \subseteq \{\alpha \mid f[P] \xrightarrow{\alpha}\}$. □

The following result summarizes the contribution of our work: the adoption of filters enlarges the number of services that satisfy a client.

Corollary 4.7. *If* $\vdash P : \rho$, $\vdash Q : \sigma$, *and* $\rho \dashv f(\sigma)$, *then* $P \dashv f[Q]$.

5. PRACTICE OF CONTRACTS

Hitherto we developed our theory by working on infinite trees, for both types and filters. The main advantage of this approach is that the resulting theory does not depend on a particular concrete syntax used to finitely denote infinite trees. Of course, the use of infinite trees is infeasible in practice, and as soon as one wants to devise typing systems and algorithms for actual languages (or just process calculi) it is necessary to introduce a concrete finite syntax to denote possibly infinite trees. Remarkably, the results stated for infinite trees easily carry over to whatever (reasonable) concrete syntax we choose to denote them, by using classic techniques dating back to Bruno Courcelle's seminal work [Courcelle 1983]. This contrasts with the fact that transposing the results stated for one syntax onto another can be quite hard, since one has to sieve the properties that hold for infinite trees from those that are meaningful only for the particular syntax at issue, whence the interest of our approach.

Choosing a particular concrete syntax neither is without consequences nor is it just a matter of taste. Two concrete syntaxes are quite popular in the process

algebras literature: the first one uses explicit recursion variables while the second uses the Kleene star to denote an unbound number of occurrences of some subtree. Each of them fits different applications. In the rest of this section we first introduce these two concrete syntaxes and we show how the main properties we studied in the previous sections for infinite trees (compliance, subcontracting, ...) transpose to them. Next we apply each concrete syntax to a Web service description language: we show that WSCL diagrams (§2.2) can be straightforwardly encoded by resorting to recursion variables and that WS-BPEL so-called activities can be naturally typed using Kleene notation. In this latter case, we define a contract based type system and, above all, show how to use filters to trim down the case explosion introduced by the use of parallel compositions of activities.

> TERMINOLOGY. In the rest of this section we introduce the two concrete syntaxes for contracts and filters we hinted above: the one based on recursive variables and the other on Kleene's stars. In order not to clutter the notation, in the concrete syntax we will use the same metavariables for contracts and filters that we used for the corresponding possibly infinite terms of the previous sections. To avoid any ambiguity we will use the terminology *unfolded* contract/filter when referring to the possibly infinite terms used so far, *recursive* contract/filter for terms generated by using explicit recursion variables, and *regular* contract/filter for terms using the Kleene star. We will omit the qualifying adjectives only when no confusion can arise.

5.1 Concrete syntax for contracts

The first concrete syntax for regular contracts is the same as given in §2.1 for unfolded contracts/terms, extended with the well-known recursion operator $\mathsf{rec}\ x = \sigma$ which binds the *recursion variable* x in σ. The idea is that an occurrence of x in σ stands for the whole $\mathsf{rec}\ x = \sigma$ term. A *recursive contract* is a finite term inductively generated by the following grammar:

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

Similarly a *recursive filter* is a finite term inductively generated by the grammar

$$f \ ::= \ \mathbf{0} \mid \alpha.f \mid f \vee f \mid f \wedge f \mid \mathsf{rec}\ x = f \mid x$$

As usual we write $\mathsf{fv}(\sigma)$ and $\mathsf{bv}(\sigma)$ for denoting the free and the bound variables occurring in σ, respectively (their definition is standard); we say that σ is *closed* if $\mathsf{fv}(\sigma) = \emptyset$; we write $\sigma\{\tau/x\}$ for the contract obtained from σ by replacing every free occurrence of x with τ; we proceed similarly for filters. Using this syntax, contractivity corresponds to requiring that in a subterm $\mathsf{rec}\ x = \sigma$ (respectively, $\mathsf{rec}\ x = f$) every free occurrence of x in σ (respectively, in f) be guarded by at least one prefix. Thus we rule out terms such as e.g. $\mathsf{rec}\ x = x + x$ or $\mathsf{rec}\ x = x \vee x$.

Intuitively, every recursive contract corresponds to the (possibly infinite) unfolded contract obtained by repeatedly unfolding every $\mathsf{rec}\ x = \sigma$ to $\sigma\{\mathsf{rec}\ x = \sigma/x\}$ (and similarly for filters). Consequently, the semantics of a term above is equal to the semantics of the infinite tree it denotes. More rigorously, let a *system of regular equations* be a finite set $\{x_1 = \sigma_1, \ldots, x_n = \sigma_n\}$ of equations where $x_1, \ldots, x_n$ are

the *unknowns* and $\sigma_1, \ldots, \sigma_n$ are recursive contracts such that $\mathsf{fv}(\sigma_i) \subseteq \{x_1, \ldots, x_n\}$ for every $1 \leq i \leq n$. We associate each closed recursive contract with a pair (E, x) where E is a system of regular equations and x one of its unknowns called the *initial unknown*. If the σ_i's of E are such that every free occurrence of a variable is guarded by at least a prefix, then the system of equations satisfies the so-called Greibach condition [Courcelle 1983] and, by Theorem 4.3.1 of [Courcelle 1983], the system admits a unique solution of unfolded contracts $(\tau_1, \ldots, \tau_n)$ such that $\tau_i = \sigma_i\{\tau_1/x_1\} \cdots \{\tau_n/x_n\}$ for every $1 \leq i \leq n$. Then, we define the semantics of a closed recursive contract as the τ_i component corresponding to the initial unknown associated with it.

Let σ be a closed recursive contract such that $\mathsf{bv}(\sigma) = \{x_1, \ldots, x_n\}$. Without loss of generality, assume that every x_i is bound exactly once in σ. Let $\mathscr{E}(\sigma)$ be the function inductively defined by the rules:

$$\mathscr{E}(\mathbf{0}) = \mathbf{0} : \emptyset \qquad \mathscr{E}(x) = x : \emptyset \qquad \frac{\mathscr{E}(\sigma) = \sigma' : E}{\mathscr{E}(\alpha.\sigma) = \alpha.\sigma' : E}$$

$$\frac{\mathscr{E}(\sigma) = \sigma' : E \quad \mathscr{E}(\tau) = \tau' : E'}{\mathscr{E}(\sigma + \tau) = \sigma' + \tau' : E \cup E'} \qquad \frac{\mathscr{E}(\sigma) = \sigma' : E \quad \mathscr{E}(\tau) = \tau' : E'}{\mathscr{E}(\sigma \oplus \tau) = \sigma' \oplus \tau' : E \cup E'}$$

$$\frac{\mathscr{E}(\sigma) = \sigma' : E \quad x \notin \mathsf{fv}(\sigma)}{\mathscr{E}(\mathsf{rec}\ x = \sigma) = \sigma' : E} \qquad \frac{\mathscr{E}(\sigma) = \sigma' : E \quad x \in \mathsf{fv}(\sigma)}{\mathscr{E}(\mathsf{rec}\ x = \sigma) = x : E \cup \{x = \sigma'\}}$$

The pair composed of the system of regular equations and the initial unknown associated with σ, denoted by $\mathsf{R}(\sigma)$, is defined as follows:

$$\mathsf{R}(\sigma) \stackrel{\text{def}}{=} \begin{cases} (E, x_i) & \text{if } \mathscr{E}(\sigma) = x_i : E \\ (E \cup \{x_0 = \sigma'\}, x_0) & \text{otherwise} \end{cases}$$

The semantics $[\![\sigma]\!]$ of the recursive contract σ is the unfolded contract τ_i, where $(\tau_0, \tau_1, \ldots, \tau_n)$ is the unique solution of the system E such that $\mathsf{R}(\sigma) = (E, x_i)$.

The following proposition formalizes the fact that a recursive contract $\mathsf{rec}\ x = \sigma$ and its unfolding $\sigma\{\mathsf{rec}\ x = \sigma/x\}$ are equivalent, namely that they are associated with the same regular system and hence they denote the same regular contract.

PROPOSITION 5.1. *Let* $\mathscr{E}(\mathsf{rec}\ x = \sigma) = x : E \cup \{x = \sigma'\}$. *Then* $\mathscr{E}(\sigma\{\mathsf{rec}\ x = \sigma/x\}) = \sigma' : E \cup \{x = \sigma'\}$.

PROOF. We prove a more general statement. Let $\mathsf{dom}(E)$ be the set of unknowns in a regular system E; let $E\{\tau/x\} \stackrel{\text{def}}{=} \{y = \sigma\{\tau/x\} \mid x = \sigma \in E\}$, where we assume that $\mathsf{dom}(E) \cap \mathsf{fv}(\tau) = \emptyset$; let σ and τ be two contracts such that $\mathsf{bv}(\sigma) \cap \mathsf{fv}(\tau) = \emptyset$. We prove that if $x \in \mathsf{fv}(\sigma)$, then $\mathscr{E}(\sigma\{\tau/x\}) = \sigma'\{\tau'/x\} : E\{\tau'/x\} \cup F$, where $\mathscr{E}(\sigma) = \sigma' : E$ and $\mathscr{E}(\tau) = \tau' : F$. If this holds, the proposition follows immediately by posing $\tau = \mathsf{rec}\ x = \sigma$. Indeed, if $x \notin \mathsf{fv}(\sigma)$, then $\mathscr{E}(\tau) = \mathscr{E}(\sigma) = \mathscr{E}(\sigma\{\tau/x\})$. On the other hand, if $x \in \mathsf{fv}(\sigma)$, then $\mathscr{E}(\sigma\{\tau/x\}) = \sigma'\{x/x\} : E\{x/x\} \cup E \cup \{x = \sigma'\}$ and we conclude immediately since $\sigma'\{x/x\} \equiv \sigma'$ and $E\{x/x\} = E$.

As regards the more general statement, we prove it by induction on σ (recall that σ is a recursive contract, hence it is finite).

($\sigma \equiv \mathbf{0}$). Trivial since $x \notin \mathsf{fv}(\sigma)$.

($\sigma \equiv x$). We have $\sigma' \equiv x$ and $E = \emptyset$. Now $\mathscr{E}(\sigma\{\tau/x\}) = \mathscr{E}(\tau) = \tau' : F$ and we conclude by observing that $\sigma'\{\tau'/x\} \equiv \tau'$ and $E\{\tau'/x\} = \emptyset$.

($\sigma \equiv y$, $y \neq x$). Trivial since $x \notin \mathsf{fv}(\sigma)$.

($\sigma \equiv \alpha.\sigma_1$). We have $\sigma' \equiv \alpha.\sigma_1'$, where $\mathscr{E}(\sigma_1) = \sigma_1' : E$. From $x \in \mathsf{fv}(\sigma)$ we deduce $x \in \mathsf{fv}(\sigma_1)$, hence by induction hypothesis we derive $\mathscr{E}(\sigma_1\{\tau/x\}) = \sigma_1'\{\tau'/x\} : E\{\tau'/x\} \cup F$. Now $\mathscr{E}(\sigma\{\tau/x\}) = \mathscr{E}(\alpha.\sigma_1\{\tau/x\}) = \alpha.\sigma_1'\{\tau'/x\} : E\{\tau'/x\} \cup F$ and we conclude by observing that $\sigma'\{\tau'/x\} \equiv \alpha.\sigma_1'\{\tau'/x\}$.

($\sigma \equiv \sigma_1 + \sigma_2$). We have $\sigma' \equiv \sigma_1' + \sigma_2'$ and $E = E_1 \cup E_2$, where $\mathscr{E}(\sigma_1) = \sigma_1' : E_1$ and $\mathscr{E}(\sigma_2) = \sigma_2' : E_2$. We examine only one interesting case, when $x \in \mathsf{fv}(\sigma_1) \setminus \mathsf{fv}(\sigma_2)$. By induction hypothesis we derive $\mathscr{E}(\sigma_1\{\tau/x\}) = \sigma_1'\{\tau'/x\} : E_1\{\tau'/x\} \cup F$. Now $\mathscr{E}(\sigma\{\tau/x\}) = \mathscr{E}(\sigma_1\{\tau/x\} + \sigma_2) = \sigma_1'\{\tau'/x\} + \sigma_2' : E_1\{\tau'/x\} \cup E_2 \cup F$ and we conclude by observing that $\sigma'\{\tau'/x\} \equiv \sigma_1'\{\tau'/x\} + \sigma_2'$ and $E\{\tau'/x\} = E_1\{\tau'/x\} \cup E_2$.

($\sigma \equiv \sigma_1 \oplus \sigma_2$). Similar to the previous case.

($\sigma \equiv \mathsf{rec}\ x = \sigma_1$). Trivial since $x \notin \mathsf{fv}(\sigma)$.

($\sigma \equiv \mathsf{rec}\ y = \sigma_1, y \neq x$). If $x \in \mathsf{fv}(\sigma)$, then $x \in \mathsf{fv}(\sigma_1)$ because $x \neq y$. We distinguish two subcases. Assume $y \notin \mathsf{fv}(\sigma_1)$. Then $\mathscr{E}(\sigma) = \mathscr{E}(\sigma_1) = \sigma' : E$. By induction hypothesis we derive $\mathscr{E}(\sigma_1\{\tau/x\}) = \sigma'\{\tau'/x\} : E\{\tau'/x\} \cup F$. From $\mathsf{bv}(\sigma) \cap \mathsf{fv}(\tau) = \emptyset$ we obtain $y \notin \mathsf{fv}(\tau)$, hence we conclude $\mathscr{E}(\sigma\{\tau/x\}) = \mathscr{E}(\mathsf{rec}\ y = \sigma_1\{\tau/x\}) = \sigma'\{\tau'/x\} : E\{\tau'/x\} \cup F$. Assume $y \in \mathsf{fv}(\sigma_1)$. Then $\sigma' \equiv y$ and $E = E' \cup \{y = \sigma_1'\}$ where $\mathscr{E}(\sigma_1) = \sigma_1' : E'$. By induction hypothesis we derive $\mathscr{E}(\sigma_1\{\tau/x\}) = \sigma_1'\{\tau'/x\} : E'\{\tau'/x\} \cup F$. Now $\mathscr{E}(\sigma\{\tau/x\}) = \mathscr{E}(\mathsf{rec}\ y = \sigma_1\{\tau/x\}) = y : E'\{\tau'/x\} \cup F \cup \{y = \sigma_1'\{\tau'/x\}\}$ and we conclude by observing that $\sigma'\{\tau'/x\} \equiv y$ and $E\{\tau'/x\} = E'\{\tau'/x\} \cup \{y = \sigma_1'\{\tau'/x\}\}$. □

Now that we have defined the semantics of a recursive contract in terms of unfolded contracts, we can straightforwardly extend to recursive contracts all the definitions introduced for recursive contracts. For example, if σ is a recursive contract, then $\sigma \Downarrow \mathrm{R}$ if and only if $[\![\sigma]\!] \Downarrow \mathrm{R}$.

By Theorem 4.2.1 of [Courcelle 1983] we also know that the syntax of recursive contracts is complete with respect to the set of unfolded contracts. In particular, every unfolded contract is a component of the unique solution of some regular system. A regular system is nothing but a flattened recursive contract, in which every recursion has been turned into an equation.

The second syntax we consider is reminiscent of regular expressions, with the remarkable difference that we have two different sum operators $+$ and $\oplus$ for external and internal choice, respectively. Correspondingly we also have two different Kleene star operators $*$ and $\circledast$. This yields to the definition of *regular* contracts and filters as the finite terms inductively generated by the following grammars

$$\begin{array}{rcl} \sigma & ::= & \mathbf{0} \mid \alpha \mid \sigma;\sigma \mid \sigma + \sigma \mid \sigma \oplus \sigma \mid \sigma^* \mid \sigma^{\circledast} \\ f & ::= & \mathbf{0} \mid \alpha \mid f;f \mid f \vee f \mid f \wedge f \mid f^* \end{array}$$

The semantics of a regular contract τ can be indirectly given by translating it into a recursive contract. The main issue of this translation is handling sequential composition, which must be reduced to action prefixes. We parametrize the translation with a *continuation* on which we accumulate actions while scanning

the regular contract from right to left. We write $[\![\tau]\!]_\sigma$ for the recursive contract resulting from the encoding of the regular contract τ, when the continuation is the recursive contract σ. We must ensure that the translation does not yield degenerate recursive contracts in which some bound variable occurs unguarded, since such terms do not have a proper semantics. To this purpose we inductively define a guardedness predicate $\mathsf{G}(\sigma)$ guaranteeing that in the recursive contract $[\![\sigma]\!]_x$ the variable x always occurs under a prefix. The $\mathsf{G}(\sigma)$ predicate is inductively defined as follows:

—$\mathsf{G}(\mathbf{0})$ and $\mathsf{G}(\alpha)$;
—if $\mathsf{G}(\sigma)$ or $\mathsf{G}(\tau)$, then $\mathsf{G}(\sigma;\tau)$;
—if $\mathsf{G}(\sigma)$ and $\mathsf{G}(\tau)$, then $\mathsf{G}(\sigma+\tau)$ and $\mathsf{G}(\sigma\oplus\tau)$.

and $[\![\tau]\!]_\sigma$ is inductively defined thus:

$$\begin{array}{rcll}
[\![\mathbf{0}]\!]_\sigma &=& \mathbf{0} & \\
[\![\alpha]\!]_\sigma &=& \alpha.\sigma & \\
[\![\tau;\tau']\!]_\sigma &=& [\![\tau]\!]_{[\![\tau']\!]_\sigma} & \\
[\![\tau+\tau']\!]_\sigma &=& [\![\tau]\!]_\sigma + [\![\tau']\!]_\sigma & \\
[\![\tau\oplus\tau']\!]_\sigma &=& [\![\tau]\!]_\sigma \oplus [\![\tau']\!]_\sigma & \\
[\![\tau^*]\!]_\sigma &=& \mathsf{rec}\ x = \sigma + [\![\tau]\!]_x & (x \text{ fresh and } \mathsf{G}(\tau)) \\
[\![\tau^{\circledast}]\!]_\sigma &=& \mathsf{rec}\ x = \sigma \oplus [\![\tau]\!]_x & (x \text{ fresh and } \mathsf{G}(\tau))
\end{array}$$

We write $[\![\tau]\!]$ for $[\![\tau]\!]_{\mathbf{0}}$. Note that neither $[\![\tau^*]\!]_\sigma$ nor $[\![\tau^{\circledast}]\!]_\sigma$ are defined if x occurs unguarded in $[\![\tau]\!]_x$. Similarly to what happens to the external choice operator $+$, the external Kleene star may hide an internal Kleene star if the contract being iterated shares an action with the continuation. For instance,

$$[\![\alpha^*]\!]_\alpha = \mathsf{rec}\ x = \alpha + \alpha.x \simeq \mathsf{rec}\ x = \alpha \oplus \alpha.x = [\![\alpha^{\circledast}]\!]_\alpha$$

It is well known that if one relies on a nondeterministic interpretation of regular expressions (which leads to dropping the left distributivity law: $X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$), then these are not complete with respect to nondeterministic finite state automata, in the sense that there are languages recognized by nondeterministic finite state automata that cannot be generated by regular expressions [De Nicola and Labella 2003]. In our context this means that there are recursive contracts for which there exists no regular contract having an equivalent unfolding. For instance, there is no regular contract whose translation is the recursive contract $\mathsf{rec}\ x = (a.x + b) \oplus (c.x + d)$. Roughly speaking, the reason lies in the fact that the same recursion variable x is shared among two different "loops" in the recursive contract. Characterizations of recursive contracts admitting equivalent regular contracts can be found in [De Nicola and Labella 2003; Baeten et al. 2007].

5.2 Encoding WSCL activity diagrams

A WSCL activity diagram [Banerji et al. 2002] is a tuple $(Q, \mathrm{A}, \delta, x_1, x_n)$ where $Q = \{x_1, \ldots, x_n\}$ is a finite set of *interactions*, $\mathrm{A} = \{\alpha_1, \ldots, \alpha_m\}$ is a finite set of actions representing ingoing and outgoing *document types*, $\delta \subseteq Q \times \mathrm{A} \times Q$ is the *transition relation*, $x_1 \in Q$ is the *initial interaction* and $x_n \in Q$ is the *final interaction*. We write $x \xrightarrow{\alpha} y$ if $(x, \alpha, y) \in \delta$; we write $x \xrightarrow{\alpha}$ if there exists $y \in Q$ such that $x \xrightarrow{\alpha} y$; we write $x \xrightarrow{\alpha}\!\!\!\!\!/\ $ if not $x \xrightarrow{\alpha}$.

According to the WSCL specification, the following well-formedness conditions must hold:

(1) every interaction must be reachable from x_1, namely for every $y \in Q$ and $y \neq x_1$ there exist $\alpha_1, \ldots, \alpha_k$ such that $x_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_k} y$;
(2) for every $x \in Q$, if there exists $a \in \text{A}$ such that $x \xrightarrow{a}$, then for every $\overline{b} \in \text{A}$ we have that $x \xrightarrow{\overline{b}}$;
(3) x_n must not have any outgoing transition, that is $x_n \xrightarrow{\alpha}$ for every $\alpha \in \text{A}$.

The encoding of an interaction x in a WSCL activity diagram when the interactions Q have not been encoded yet is denoted by $E(x, Q)$ and is defined as follows:

$$E(x,Q) \stackrel{\text{def}}{=} \begin{cases} x & \text{if } x \notin Q \\ \mathsf{rec}\ x = \bigoplus_{\overline{a} \in \text{A}, \delta(x,\overline{a}) \neq \emptyset} \bigoplus_{y \in \delta(x,\overline{a})} \overline{a}.E(y, Q \setminus \{x\}) & \\ \qquad \text{if } x \in Q \text{ and } x \xrightarrow{\overline{a}} \text{ for some } \overline{a} \in \text{A} & \\ \mathsf{rec}\ x = \sum_{a \in \text{A}, \delta(x,a) \neq \emptyset} \sum_{y \in \delta(x,a)} a.E(y, Q \setminus \{x\}) & \text{otherwise} \end{cases}$$

Because of well-formedness condition (1), the whole activity diagram is encoded; because of well-formedness condition (2), the last two cases in the definition of $E(x, Q)$ are mutually exclusive; because of well-formedness condition (3), we have $E(x_n, Q) \simeq \mathbf{0}$. The encoding of a WSCL activity diagram is now defined as $E(x_1, Q)$.

Checking that the WSCL activity diagram of Figure 1 yields the recursive definitions given in §2.2 is an exercise as straightforward as tedious. Equally straightforward is to deduce the encoding of the "1-click ordering" extension and of the corresponding filter defined at the end of §3.2.1. That example shows that filters are needed even when service behavior is described by WSCL diagrams, which restrict external choices to output actions and internal choices to input actions (see Remark 5.5 for a thorough analysis).

5.3 Typing WS-BPEL activities

WS-BPEL [Alves et al. 2007] is an OASIS standard language for the description of business processes. It builds on top of standard Web service technologies, such as WSDL, for providing a detailed, structured description of Web services behavior, including exception and compensation handlers. Being a concrete Web service specification language, WS-BPEL is hard to formalize thoroughly. In our setting, however, we are merely concerned with the observable behavior of a Web service, hence we can disregard any detail that is not directly related with the interactions of the Web service with the external world. Furthermore, we gain in clarity by getting rid of the heavyweight XML syntax used in WS-BPEL and by preferring a streamlined algebraic presentation, which the reader will easily match with the original language.

$$\begin{array}{rcll} A & ::= & \mathtt{action}\langle\alpha\rangle & \\ & | & \mathtt{pick}\langle a_1.A_1, \ldots, a_n.A_n\rangle & n \geq 0 \\ & | & \mathtt{sequence}\langle A_1, \ldots, A_n\rangle & n \geq 0 \\ & | & \mathtt{if}\langle A_1, \ldots, A_n\rangle & n \geq 1 \\ & | & \mathtt{flow}\langle A_1, \ldots, A_n\rangle & n \geq 0 \\ & | & \mathtt{while}\langle A\rangle & \end{array}$$

Table IV. Type system for WS-BPEL activities.

$$\text{(ACTION)}\quad \vdash \mathtt{action}\langle\alpha\rangle : \alpha \qquad \text{(PICK)}\quad \frac{\vdash A_1 : \sigma_1 \quad \cdots \quad \vdash A_n : \sigma_n}{\vdash \mathtt{pick}\langle a_1.A_1, \ldots, a_n.A_n\rangle : a_1;\sigma_i + \cdots + a_n;\sigma_n}$$

$$\text{(SEQUENCE)}\quad \frac{\vdash A_1 : \sigma_1 \quad \cdots \quad \vdash A_n : \sigma_n}{\vdash \mathtt{sequence}\langle A_1, \ldots, A_n\rangle : \sigma_1;\ldots;\sigma_n} \qquad \text{(IF)}\quad \frac{\vdash A_1 : \sigma_1 \quad \cdots \quad \vdash A_n : \sigma_n}{\vdash \mathtt{if}\langle A_1, \ldots, A_n\rangle : \sigma_1 \oplus \cdots \oplus \sigma_n}$$

$$\text{(FLOW)}\quad \frac{\vdash A_1 : \sigma_1 \quad \cdots \quad \vdash A_n : \sigma_n \quad \mathsf{actions}(\sigma_i) \cap \mathsf{co}(\mathsf{actions}(\sigma_j)) = \emptyset^{\ i,j\in 1..n}}{\vdash \mathtt{flow}\langle A_1, \ldots, A_n\rangle : \sigma_1 \parallel \cdots \parallel \sigma_n} \qquad \text{(WHILE)}\quad \frac{\vdash A : \sigma}{\vdash \mathtt{while}\langle A\rangle : \sigma^{\circledast}}$$

The above grammar describes a substantial fragment of WS-BPEL so-called *activities* (i.e., processes).[4] The activity $\mathtt{action}\langle\overline{a}\rangle$ denotes the invocation of operation a or, more generally, the act of sending a message on a channel identified by a; it can be used for representing both `invoke` and `reply` activities in WS-BPEL. The activity $\mathtt{action}\langle a\rangle$ denotes the act of waiting for an interaction on a channel identified by a; it can be used for representing `receive` activities. The `pick` activity allows the service to provide multiple operations, among which the client can choose which one to execute. Each action a_i denotes an operation that, when invoked, causes the corresponding activity A_i to be executed; the `sequence` activity sequentially activates the specified sub-activities, A_{i+1} starting only when A_i is completed; the `if` activity performs an internal choice among the specified sub-activities according to the result of some Boolean conditions that we leave unspecified in the syntax; the `while` activity sequentially executes the specified sub-activity as long as some Boolean condition (once more left unspecified) is verified; the `flow` denotes the parallel composition of the specified sub-activities; it completes when all the sub-activities have completed. We write `empty` for $\mathtt{sequence}\langle\rangle$, for $\mathtt{pick}\langle\rangle$, and for $\mathtt{flow}\langle\rangle$.

Table IV shows the type system for WS-BPEL activities, where a judgment $\vdash A : \sigma$ associates an activity A with its contract σ. Most rules are completely straightforward as WS-BPEL activities map very naturally to the contract operators described in §5.1. The only rules that deserve some explanation are (FLOW) and (WHILE). Regarding (WHILE), the choice as to whether the sub-activity A must be executed once more or the repetition has ended is made internally, by evaluating some unspecified condition. Hence the use of the internal Kleene star $\circledast$ in the resulting contract. Because of the side-condition in the semantics of $\sigma^{\circledast}$, not every `while` activity is well typed. For instance, $\mathtt{while}\langle\mathtt{if}\langle\mathtt{empty}, \mathtt{action}\langle\overline{a}\rangle\rangle\rangle$ is not well typed because the contract of the repeated activity has an empty ready set. Regard-

[4]We did not considered (*i*) activities, such as `throw` or `rethrow`, for signalling exceptions and handle them; (*ii*) activities, such as `repeatUntil` or `forEach`, that can be encoded in the given fragment; (*iii*) activities, such as `wait` or `assign`, whose effects are not directly observable outside the process. The WS-BPEL standard also defines a number of so-called static analysis requirements to constrain the form of activities. We disregard them since they mostly deal with aspects, such as scope and link declarations, that we want to abstract from.

ing (FLOW), the sub-activities are allowed to run concurrently and independently of each other. Thus, the contract of a `flow` activity is given by all the possible interleaving of actions in the contracts of the sub-activities. This interleaving is computed by the $\|$ operator, which is defined as follows:

$$\sigma \,\|\, \tau \stackrel{\text{def}}{=} \bigoplus_{\sigma \Downarrow \mathrm{R}, \tau \Downarrow \mathrm{S}} (\textstyle\sum_{\alpha \in \mathrm{R}} \alpha; (\sigma(\alpha) \,\|\, \tau) + \sum_{\beta \in \mathrm{S}} \beta; (\sigma \,\|\, \tau(\beta)))$$

It is easy to verify that $\sigma \,\|\, \tau$ is well defined (it is a regular contract, viz., it satisfies guardedness conditions) and that $\|$ is a commutative, associative operator whose neutral element is $\mathbf{0}$. The premise $\mathsf{actions}(\sigma_i) \cap \mathsf{co}(\mathsf{actions}(\sigma_j)) = \emptyset$ (we use $\mathsf{actions}(\sigma)$ to denote the set of all actions occurring in σ) imposes that no message exchange is allowed within the same WS-BPEL business process.[5] This allows us to express $\|$ as a simplified form of the *expansion law* as it is found in [De Nicola and Hennessy 1987; Hennessy 1988]. Briefly, the interleaving of σ and τ is equivalent to an internal choice of all possible combinations of an internal choice of σ with an internal choice of τ; for every such combination the interleaving contract gives the client the choice to synchronize either with an α action of σ (in which case it then continues with the interleaving of the continuation of α and τ) or with a β action of τ (in which case it then continues with the interleaving of the continuation of β and σ).

We do not define any operational semantics for the activities. As a matter of fact this is already given by the type system of Table IV: contracts being behavioral types, they faithfully describe the operational semantics of activities. More importantly, our theory of contracts provides us with a formal tool for reasoning about safe replacement and upgrade of WS-BPEL activities, by comparing the corresponding contracts. For instance, the depth and width subtyping properties enjoyed by $\preceq$ tell us that it is safe to replace an activity A with an activity $\mathtt{sequence}\langle A, A' \rangle$ and also that it is safe to replace $\mathtt{pick}\langle a_1.A_1, \ldots, a_m.A_m \rangle$ with $\mathtt{pick}\langle a_1.A_1, \ldots, a_n.A_n \rangle$ where $n \geq m$. In the first case, we are appending additional functionalities to some business process; in the second case, we are providing additional alternative functionalities to a business process.

The following result shows that we can also derive interesting substitution properties for `sequence` and `flow`:

PROPOSITION 5.2. *Let* $\mathsf{actions}(\sigma) \cap \mathsf{actions}(\tau) = \emptyset$. *Then* $\sigma; \tau \preceq \sigma \,\|\, \tau$.

PROOF. It is sufficient to prove that $\mathscr{W} \stackrel{\text{def}}{=} \{(\sigma, \sigma \,\|\, \tau) \mid \mathsf{actions}(\sigma) \cap \mathsf{actions}(\tau) = \emptyset\} \cup \{(\sigma; \tau, \sigma \,\|\, \tau) \mid \mathsf{actions}(\sigma) \cap \mathsf{actions}(\tau) = \emptyset\}$ is a weak subcontract relation. Let $(\sigma, \sigma \,\|\, \tau) \in \mathscr{W}$ and assume $\sigma \,\|\, \tau \Downarrow \mathrm{R}$. By definition of $\sigma \,\|\, \tau$ there exist R' and S such that $\sigma \Downarrow \mathrm{R}'$ and $\tau \Downarrow \mathrm{S}$ and $\mathrm{R} = \mathrm{R}' \cup \mathrm{S}$, from which we derive $\mathrm{R}' \subseteq \mathrm{R}$. Let $\alpha \in \mathrm{R}'$. Then $(\sigma \,\|\, \tau)(\alpha) \equiv \sigma(\alpha) \,\|\, \tau$ by definition of $\sigma \,\|\, \tau$ and we conclude $(\sigma(\alpha), \sigma(\alpha) \,\|\, \tau) \in \mathscr{W}$ by definition of $\mathscr{W}$. Let $(\sigma; \tau, \sigma \,\|\, \tau) \in \mathscr{W}$ and assume $\sigma \,\|\, \tau \Downarrow \mathrm{R}$. Then there exist R' and S such that $\sigma \Downarrow \mathrm{R}'$ and $\tau \Downarrow \mathrm{S}$ and $\mathrm{R} = \mathrm{R}' \cup \mathrm{S}$. We have two cases: (1) if $\mathrm{R}' \neq \emptyset$, then $\sigma; \tau \Downarrow \mathrm{R}'$ and we derive $\mathrm{R}' \subseteq \mathrm{R}$; (2) if $\mathrm{R}' = \emptyset$, then $\sigma; \tau \Downarrow \mathrm{S}$ and we derive $\mathrm{S} \subseteq \mathrm{R}$. In case (1), assume $\alpha \in \mathrm{R}'$. From the encoding of $\sigma; \tau$ and from the hypothesis $\mathsf{actions}(\sigma) \cap \mathsf{actions}(\tau) = \emptyset$ we have $(\sigma; \tau)(\alpha) \equiv \sigma(\alpha); \tau$ and by definition of $\|$ we

[5] In WS-BPEL, actions used for synchronizing activities of a flow must be invisible outside the flow.

have $(\sigma \parallel \tau)(\alpha) \equiv \sigma(\alpha) \parallel \tau$ and we conclude $(\sigma(\alpha); \tau, \sigma(\alpha) \parallel \tau) \in \mathscr{W}$ by definition of $\mathscr{W}$. In case (2), assume $\alpha \in \mathrm{S}$. From the encoding of $\sigma; \tau$ and from the hypothesis $\mathsf{actions}(\sigma) \cap \mathsf{actions}(\tau) = \emptyset$ we obtain $(\sigma; \tau)(\alpha) \equiv \tau(\alpha)$ and by definition of $\parallel$ we have $(\sigma \parallel \tau)(\alpha) \equiv \sigma \parallel \tau(\alpha)$ so we conclude $(\tau(\alpha), \sigma \parallel \tau(\alpha)) \in \mathscr{W}$ again by definition of $\mathscr{W}$. □

As a corollary of Proposition 5.2 observe that $\sigma \preceq \sigma \parallel \tau$ when σ and τ share no common action. Let $f : \sigma; \tau \leq \sigma \parallel \tau$ and let $\vdash A : \sigma$ and $\vdash B : \tau$. Proposition 5.2 can be interpreted in two different ways: when reading $\sigma; \tau \preceq \sigma \parallel \tau$ from left to right, the proposition gives us sufficient conditions by which we can replace $\mathtt{sequence}\langle A, B\rangle$ with $\mathtt{flow}\langle A, B\rangle$ when A and B are independent activities and we want to increase the service throughput by taking advantage of parallelism (for instance, because the machine hosting the Web service has been upgraded and is now multiprocessor). In this case the clients of the old, sequential Web service can still interact successfully with the upgraded, parallel one, provided that the interaction is shielded by f.

On the other hand, when reading $\sigma; \tau \preceq \sigma \parallel \tau$ from right to left, the proposition gives us sufficient conditions by which we can approximate the behavior of $\mathtt{flow}\langle A, B\rangle$ with that of $\mathtt{sequence}\langle A, B\rangle$. Indeed, the size of contract $\sigma \parallel \tau$ may grow exponentially with respect to the size of σ and τ because of the use of continuations $\sigma(\alpha)$ and $\tau(\alpha)$ and of the interleaving semantics in the definition of $\parallel$. For instance, we have

$$a.\overline{b} \parallel c.\overline{d} = a.(\overline{b}.c.\overline{d} + c.(\overline{b}.\overline{d} + \overline{d}.\overline{b})) + c.(a.(\overline{b}.\overline{d} + \overline{d}.\overline{b}) + \overline{d}.a.\overline{b})$$

and it might be desirable to approximate $a.\overline{b} \parallel c.\overline{d}$ with $a.c.\overline{b}.\overline{d}$. However, doing so without filters can be dangerous, as the following example shows. Consider the client contract $\rho \stackrel{\text{def}}{=} \overline{a}.(b.b.\mathtt{e} + \overline{c}.b.d.\mathtt{e})$. Then $\rho \dashv a.c.\overline{b}.\overline{d}$, hence a client having contract ρ would be declared compliant with the service whose approximate contract is $a.c.\overline{b}.\overline{d}$. However, by inspecting the expansion of $a.\overline{b} \parallel c.\overline{d}$ we realize that the contract of the service does actually provides some behaviour for the case for a $\overline{b}$ action right after the action a, so the client (process) might get stuck trying to read a second b after the first one. By applying the filter $a.c.\overline{b}.\overline{d}$, we prevent any synchronization on b to happen during the second interaction, thus guaranteeing that the client successfully terminates.

We conclude this section with some observations on one of the characteristics that distinguish our approach from the current literature on service contracts and session types and that make it more general. While in all work on contracts and sessions we are aware of external choices take place on input actions and internal ones emit output actions, in our formalism we have no such restrictions and therefore we allow input actions to guard internal choices and output actions to guard external choices. The type system in Table IV shows that both these features are necessary to type WS-BPEL processes. In particular, as regards internal choices observe that there is no constraint on the actions that can be used in an `if` activity. Thus, for instance, one can write $\mathtt{if}\langle\mathtt{action}\langle\overline{a}\rangle, \mathtt{action}\langle b\rangle\rangle$ whose type, $\overline{a} \oplus b$, performs an internal choice that may issue an input on b. As regards external choices, the activities in a `flow` activity can be of any form. Thus, for instance, the contract of the activity $\mathtt{flow}\langle\mathtt{action}\langle\overline{a}\rangle, \mathtt{action}\langle b\rangle\rangle$ turns out to be $\overline{a}.b + b.\overline{a}$ which guards one branch of an external choice with an output action $\overline{a}$.

External choices on outputs may also directly stem out from more advanced WS-BPEL usages. Indeed, consider the contract $\overline{a} \oplus b$ above corresponding to an `if` activity. A client of a service with this behavior must necessarily offer, by means of an external choice, both an input on a *and* an output on b. The simplified abstract syntax for WS-BPEL we have provided does not permit the specification of a process with this behavior, as we have imposed the restriction that all activities in a `pick` activity are guarded by the input actions. This is the most common usage for `pick` activities, since in WS-BPEL `pick` elements usually contain a list of several `onMessage` *events*. For instance the following streamlined WS-BPEL syntax

```
<pick ... >
   <onMessage operation="a_1" ... > A_1 </onMessage>
       ⋮
   <onMessage operation="a_n" ... > A_n </onMessage>
</pick>
```

denotes the term $\texttt{pick}\langle a_1.A_1, \ldots, a_n.A_n\rangle$. However WS-BPEL allows the programmer to enrich pick activities with an `onAlarm` event. This event specifies a timeout and an activity that will be executed if no `onMessage` event is received within the timeout. From the behavioural point of view, `onAlarm` events are indistinguishable from `onMessage` ones. In particular, it would be *wrong* to model timeouts with internal choices, as this would make the process incompatible with those parties that do readily send a message targeted to one of the `onMessage` events. Since there is no restriction on the activity of an `onAlarm` event, then this can be an output activity. For instance, the activity

```
<pick>
   <onMessage operation="Invoice"> A_1 </onMessage>
   <onAlarm>
     <for>'P3DT10H'</for>        <!-- timeout interval of 3 days and 10 hours -->
     <sequence> <invoke operation="RequestInvoice" ... > A_2 </sequence>
   </onAlarm>
</pick>
```

would be typed as $\texttt{Invoice}.A_1 + \overline{\texttt{RequestInvoice}}.A_2$, thus mixing once more input and output actions in an external choice.

5.4 On implementing filters

We have presented filters as behavioral coercions that enlarge the set of Web services a client is compliant with. Basically, the filter mediates the interaction between client and service by forbidding actions that can potentially lead the client to a deadlock. The filter, however, has no power over the internal choices made independently by client and service. In our theory, this latter fact is remarkably rendered by rules (ICHOICE) and (ECHOICE) (see Table I on page 21), in which the *same* filter must be applicable for all the branches of internal and external choices, in order to work correctly. Do filters actually play a role in real-world Web services? If so, do they admit effective, and hopefully efficient, implementations? As regards the first question, we notice that a filter is actually a well known actor in the Web service scenario: it is an example of Web service *orchestrator*. An orchestrator is

simply a process whose task is to coordinate other processes in such a way so as to guarantee that their interaction eventually leads to the achievement of a goal (in our setting, the goal being client satisfaction).

As regards the implementation of filters, we discuss it in the rest of this section. Before moving on, let us stress that the purpose of what follows is not to show the use, importance, and/or expressiveness of filters but, rather, what their implementation could possibly look like in some well-known frameworks and reasonable usage scenarios. In other terms we show that given an existing set of clients and services, it is possible to use well-known distributed abstractions (such as join patterns) and/or current technology (such as WS-BPEL processes) to implement filters that enhance compatibility in this set.

5.4.1 *Implementing filters as join patterns.* A first observation is that filters can be rendered by means of *binary join patterns* (in the style of the Join calculus [Fournet and Gonthier 1996; Fournet et al. 1996]) of the form $\alpha \,\&\, \beta$, which can be thought of as an atomic action that can be executed provided that $\overline{\alpha}$ and $\overline{\beta}$ are simultaneously available in the execution environment. Then, a filter f may be implemented as a (finite-state) process $\mathsf{C}[\![f]\!]$ as follows

$$\mathsf{C}[\![f]\!] = \textstyle\sum_{f \overset{\alpha}{\longmapsto} f'} \alpha \,\&\, \overline{\alpha}'.\mathsf{C}[\![f']\!]$$

where we use α' to distinguish the action α performed by the filtered service, so that it is not confused with the action α as it is seen by the client. Then a client P and a service Q safely interact with each other under the supervision of the filter/orchestrator:

$$P \parallel \mathsf{C}[\![f]\!] \parallel Q\{\alpha'_1/\alpha_1, \ldots, \alpha'_n/\alpha_n\}$$

where $\alpha_1, \ldots, \alpha_n$ are the actions occurring in Q and $\parallel$ denotes the usual parallel composition of processes.

The problem then reduces to the effective implementation of join patterns. It is well known that implementations of the Join calculus all pose strong requirements over the channels being joined together, namely that they must all be created simultaneously with the patterns in which they are involved, and that they must all be local to the host that created them. However, it has also been shown in [Laneve and Padovani 2006] how to partly relax these constraints so that only locality is actually required in order to avoid a global consensus problem, and that join patterns can be dynamically attached to the site hosting the channels being joined.

5.4.2 *Filters as one-position buffers.* The second observation is that, for an interesting subclass of processes, join patterns are not necessary at all. If we assume that, at any time, the service can only (internally) decide to send messages, or can only (externally) wait for messages, then the compilation scheme sketched above reduces to the straightforward process

$$\mathsf{C}[\![f]\!] = \textstyle\sum_{f \overset{a}{\longmapsto} f'} a.\overline{a}'.\mathsf{C}[\![f']\!] + \sum_{f \overset{\overline{a}}{\longmapsto} f'} a'.\overline{a}.\mathsf{C}[\![f']\!]$$

in which, for every f, at most one of the two sums is nonempty. This subclass is interesting because it exactly characterizes processes behaving according to *session types* [Honda 1993; Takeuchi et al. 1994; Honda et al. 1998] (as well as to other theories of contracts proposed in literature: *cf.* §6 on related work) where at any

time only one process, either the client or the service, decides the next action to be executed. In this respect, the generality of our contract language might at first appear excessive, since it allows a mixture of input and output actions irrespective of the choice operators with which they are combined. In particular, at first glance it might seem reasonable—and even natural—to restrict output actions to internal choices, and input actions to external choices. However, it is not so: as it is clearly shown by Table IV, by the definition of the "$\|$" operator given earlier, and by the examples that close §5.3 such a generality is necessary in practice, since input and output actions can occur as guards in both internal and external choices of real world Web services contracts.

5.4.3 *Implementing filters in* WS-BPEL. The language of filters we have adopted is very simple. Given $f : \sigma \leq \tau$, the structure of f tells us very little about the actual behavior of σ and τ. It is reasonable to expect that, by enriching the language of filters with constructs that more faithfully describe the structure of the *proof* that relates σ and τ, one is able to provide efficient implementations in a wider range of situations. Nevertheless even if we want to implement just these "simple" filters by means of WS-BPEL processes, we cannot hope to obtain much more than what we did with one-position buffers. The point of the encoding in §5.4.1 is to exploit join patterns for detecting the simultaneous presence, on the server and on its client, of an action and its coaction. This cannot be expressed (at least not straightforwardly) in a CCS-like formalism nor in WS-BPEL, which solely relies on low-level communication primitives. Consequently we must restrict our attempt of encoding filters as WS-BPEL processes to staged computations in which just one of the partners has the floor (i.e., it has the exclusive right to fire a write action / to send a message). This, combined with the fact that the syntax of `pick` forces external choices to be performed on inputs, yields to a subcontracting hierarchy very close to the one relating session types [Gay and Hole 2005].

In what follows we outline a formalization of this hierarchy and its benefits in WS-BPEL. The point is not to study the expressiveness of WS-BPEL but to show that the theory presented in this paper can be applied to WS-BPEL and to WS-BPEL processes in the current form. In a nutshell, if we want to check whether two WS-BPEL processes are or can be made compliant, then what we have to do is (*i*) to extract their contracts by means of the type system in Table IV, (*ii*) to check whether the extracted contracts are compatible with a staged computation (that is, whether the filter that makes them compatible is an *input filter*: see Definitions 5.3 and 5.4 below), and (*iii*) to automatically synthesize the WS-BPEL process implementing the filter that makes the client compliant with the service. Thus the theory is immediately applicable to a class of existing WS-BPEL processes to combine them more flexibly (thanks to the width subtyping that allows us to update services with new features) without the need of adding new primitives or constructors (such as session types labels and labeled selections) to WS-BPEL nor of modifying the code of existing processes.

More precisely, the filters that we can reasonably encode in WS-BPEL are those that never filter out output messages and that relate staged contracts, having the following form.

DEFINITION 5.3 (STAGED CONTRACT). *We say that the (service) contract σ is*

staged *if either (1)* $\sigma \simeq \sum_{i\in I} a_i;\sigma_i$ *and for every* $i,j \in I$ *such that* $i \neq j$ *we have* $a_i \neq a_j$ *and each* σ_i *is staged or (2)* $\sigma \simeq \bigoplus_{i\in I}(\overline{a}_{i1} \parallel \cdots \parallel \overline{a}_{in_i});\sigma_i$ *and for every* $i,j \in I$ *such that* $i \neq j$ *we have* $a_{i1} \neq a_{j1}$ *and each* σ_i *is staged.*

In case (1) we say that σ is staged if it describes a service that lets the client decide which message to send. In case (2) we say that σ is staged if it describes a service that can be in one of $|I|$ different states and in each state $i \in I$ the service sends a message a_{i1} that distinguishes that state from all the others, along with possibly more messages $a_{i2}, \ldots, a_{in_i}$ (observe that $|I| > 1$ implies $n_i > 0$ for every $i \in I$). Then, the service continues behaving as a staged contract σ_i.

A filter that never filters out output messages is called *input filter* and is formally defined below.

Definition 5.4 (input filter). *We say that* f *is an* input filter *for the (service) contract* σ *if* $\sigma \overset{\varphi\overline{a}}{\longmapsto}$ *and* $f \overset{\varphi}{\longmapsto} f'$ *implies* $f' \overset{\overline{a}}{\longmapsto}$.

Namely, an input filter never filters out any output action, unless this is guarded by an input action that has been filtered out. Given an input filter f for a staged (service) contract σ, the WS-BPEL process that implements f, denoted by $F(f,\sigma)$, can be defined as follows:

$$F(f,\sigma) \stackrel{\text{def}}{=} \begin{cases} \texttt{pick}\langle a_i.\texttt{action}\langle\overline{a}'_i\rangle; F(f(a_i),\sigma_i),\ldots\rangle^{(i\in I, f\overset{a_i}{\longmapsto})} & \text{if } \sigma \simeq \sum_{i\in I} a_i.\sigma_i \\ \texttt{pick}\langle a'_{i1}.\texttt{sequence}\langle\texttt{action}\langle a'_{i2}\rangle,\ldots,\texttt{action}\langle a'_{in_i}\rangle, & \\ \qquad \texttt{flow}\langle\texttt{action}\langle\overline{a}_{i1}\rangle,\ldots,\texttt{action}\langle\overline{a}_{in_i}\rangle\rangle, & \\ \qquad F(f(\overline{a}_{i1}\cdots\overline{a}_{in_i}),\sigma_i)\rangle,\ldots\rangle^{(i\in I)} & \\ \qquad\qquad \text{if } \sigma \simeq \bigoplus_{i\in I}(\overline{a}_{i1} \parallel \cdots \parallel \overline{a}_{in_i});\sigma_i \end{cases}$$

If σ is an external choice of input actions, then the filter simply waits for a message on one of those input actions a_i that have not been filtered out ($f \overset{a_i}{\longmapsto}$). Once such a message is received, the filter delivers it to the service. If σ is an internal choice of (possibly concurrent) output actions, then the filter waits for a message a_{i1} from the service. Since σ is staged, all the a_{i1}'s are distinct hence the `pick` activity is well formed. When a message is received, the filter unambiguously knows the state the service is in, hence it collects all the other messages produced by the service in that state. Then, all the collected messages are delivered to the client, because f is an input filter.

Remark 5.5. *The reader may wonder whether one still needs filters, given that we are now considering only services that implement staged contracts. Indeed the reader may legitimately suspect that by forcing services to behave according to staged contracts we are falling back to a scenario similar to the one of sessions types, where filters are not needed. The point is that, while we assume* services *to implement staged contracts, we do not impose the same requirement for* clients*, which are free to implement any behavior that can be expressed with our contract language, including those for which filters are needed. The example described at the end of §2.2.2, showing that filters are necessary to make* WSCL*-specified services compatible, applies here as well. Indeed, notice that every contract described by a* WSCL *activity is trivially staged (for all* $i \in I$ *we have* $n_i = 1$*, in Definition 5.3) which*

means that the filter for "1-click ordering" extension can be implemented by a WS-BPEL *process and that the same problematic client that disrupts compatibility there, does the same here. It is not a coincidence that such client is not staged since it performs an external choice on the* output *of* `Buy`*. Conversely, had we considered a scenario that, besides staged services, included* only *clients with staged contracts (e.g., clients whose behavior is described by* WSCL *diagrams), then filters would have been unnecessary, as it is the case for session types.*

6. RELATED WORK

The contracts used in this presentation draw their inspiration from De Nicola and Hennessy's seminal work "CCS without $\boldsymbol{\tau}$'s" [De Nicola and Hennessy 1987], as well as from acceptance trees [Hennessy 1985; 1988] of which they can be considered an alternative representation. The works that are most closely related to ours are by Carpineti et al. [2006], Derrick et al. [1996], and those on *session types*, especially the one by Gay and Hole [2005].

In [Carpineti et al. 2006] the subcontract relation exhibits all the desirable properties we illustrated in the Introduction (§1) till equation (1), but subcontracting stops there at the problem of transitivity. In that work compliance was a syntactic notion and contracts lacked a semantic characterization. Derrick et al. [1996] provide a thorough overview of refinement relations in the testing framework that date back to the LOTOS system [Brinksma et al. 1995]. According to the terminology of [Derrick et al. 1996], the relation $\overline{a} \oplus \overline{b}.c \preceq \overline{a}$ is an instance of so-called *reduction refinement*, in which $\overline{a} \oplus \overline{b}.c$ is replaced by $\overline{a}$ thus reducing nondeterminism. On the other hand, $\overline{a} \preceq \overline{a} + \overline{b}.d$ is an instance of so-called *extension refinement*, in which a $\overline{a}$ is replaced by $\overline{a} + \overline{b}.d$ which provides further functionalities. The combination of these two refinement relations yields the so-called *implementation refinement*, which basically coincides both with the subcontract relation defined in [Carpineti et al. 2006] and with the $\ltimes$ relation we introduced in this work (see equation (2) in §2.3 and the proof of Theorem 3.4). It is known that extension refinement is not a precongruence with respect to the contract operators and that implementation refinement lacks transitivity [Derrick et al. 1996]. As already anticipated in the Introduction (§1) and formally shown in §3.3, the present paper addresses and solves both problems: precongruence can be regained under minimal conditions, namely when filtering does not depend on the internal choices of client and service and transitivity stems directly from the ability of composing filters.

Session types were introduced in the context of the π-calculus [Honda 1993; Takeuchi et al. 1994; Honda et al. 1998]. These are used to type special channels through which several messages of different types 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. With respect to our work there are two fundamental differences: the first difference is conceptual, in that session types are used for typing channels, whereas contracts are used for typing processes. Clearly a contract can be used for describing the communications occurring on the channels owned by a process, but it also describes the temporal dependencies between these communications. On the other hand, session types

describe the communications occurring on the channels in isolation. Encodings between the two formalisms are possible under some conditions as described in [Laneve and Padovani 2008]. The second difference is that 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 session types has been studied by Gay and Hole [2005], 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. However, this looks like a reasonable relation, inasmuch as $a \oplus b$ models a scenario where exactly one of two resources a and b is available (and the client does not know *which* one), which can be safely related with (and replaced by) a scenario where both a and b are available and the client can choose whether to use a or b. The example given in the Introduction (§1) right before the outline of the presentation (§1.1) is just an instance of this scenario.

Carbone et al. [Carbone et al. 2007a; 2007b] describe choreographies of Web services by means of a global calculus, and descriptions of individual processes are obtained as projections of the global description. Both the global description and the projections are based on session types. In our approach, the typical application is searching for a service compatible with a given protocol *from the client's point of view*: in particular, we want depth subtyping (a service that tries to pursue the interaction after the client has successfully terminated is compatible with this client), which does not hold for session types. In summary, we believe that our theory is more abstract than that of session types, basically because in contracts there is a neat separation of control (which is expressed as a combination of internal and external choices) and communication (which is expressed by means of actions). This allows us to type arbitrary processes, as opposed to communications in which the end-points follow a rigid discipline, and to describe more abstract synchronization patterns, such as those arising when two external choices interact with each other. Therefore, not only, as described in [Laneve and Padovani 2008], session types can be seen as a low-level implementation of the communications described by a contract, but also it is possible to draw inspiration from out theory of contracts to generalize current presentations of session types as shown by Castagna et al. [2008].

Although the work presented here considers dyadic interactions between a single client and a single server, the theory can be smoothly extended to multi-party interactions. This is proved by Bernardi et al. [2008] who extend our work (more precisely the version in [Castagna et al. 2008]) for dealing with choreographies of Web services. Apart from a few technical differences—Bernardi et al. disallow different summands of an external choice guarded by the same action and uses a fair variant of compliance—[Bernardi et al. 2008] shows how filters provide a new interesting solution to the problem of Web service composition [Traverso and Pistore 2004; Berardi et al. 2003; Hull et al. 2003; Pistore et al. 2005; De Giacomo and Sardiña

2007] in particular for what concerns the automatic synthesis of adapters [Berardi et al. 2003; De Giacomo and Sardiña 2007]. In [Bernardi et al. 2008] both contracts and filters are associated with locations and can be composed for describing and constraining the participants interacting in a choreography. The notion of compliance is extended to compositions of located contracts and an algorithm is defined that infers the greatest relevant composite filter (*cf.* §3.4.1 in this work) that makes a composition of located contracts satisfy compliance (which corresponds to our Corollary 3.24). Additionally Bernardi et al. [2008] propose a new usage of filters, which can be seen as a specification of the roles occupied by the participants of a choreography.

Since our work is about characterizing processes whose composition always successfully terminates, there is an obvious connection with the research work on termination and deadlock in process algebras. Among the vast literature on these topics, we want to single out and discuss the work by Fournet et al. [2004], as it allows us to identify the points in common but, above all, the different perspective that distinguishes the two researches. Fournet et al. [2004] define a *conformance* preorder on CCS processes with the property that a process is *stuck-free* (i.e., it successfully terminates) in every context in which smaller processes are stuck-free. The *conformance* relation of [Fournet et al. 2004] differs from our subcontract relation in some technical aspects. For example, in [Fournet et al. 2004] $a \oplus \mathbf{0} \preceq \mathbf{0}$, but $a \oplus \mathbf{0} \not\preceq a$. This essentially derives from the fact that stuck-free conformance is defined without using an explicit action (denoted by e in our work) expressing in an observationally visible way the successful termination of a party, but instead by requiring that the party must eventually reduce to the idle process $\mathbf{0}$. Doing so prevents the specification of clients of the form $\mathsf{e}+\overline{a}.\mathsf{e}$, that *attempt* to do an action, but that can succeed even if the action is not available. The lack of the explicit action e has overall important consequences on the precongruence properties of $\preceq$. A more substantial difference between our work and [Fournet et al. 2004] is the viewpoint from which processes/contracts are observed. We insist on characterizing the externally observable behavior of a service (we consider services as black boxes whose internal details do not transpire) and this has important consequences at three different levels:

(1) At the contract language level, we may disregard the internal implementation of the service, provided that the contract language is expressive enough to fully capture its behavior. Thus, Fournet et al. [2004] take into account restriction and parallel composition, while we do not since, as we explain in detail in the Conclusion (§7), these describe some internal structure of the service that we want to abstract from in its contract.

(2) At the operational semantics level, we define an "objective" transition relation of service contracts that takes into account the point of view of clients of the service. On the contrary, Fournet et al. [2004] work with the standard, "subjective" transition relation where the visible behavior of processes is given from the perspective of the process itself. A paradigmatic example that sheds light on this difference is the reduction of the process/contract $a.\sigma + a.\tau$. In our approach we have a single reduction $a.\sigma + a.\tau \stackrel{a}{\longmapsto} \sigma \oplus \tau$ (*cf.* Definition 2.2), since an external observer that has just observed the a action cannot tell whether

the observed process has taken the right or the left branch: it must consider both possibilities as an internal choice of the observed process. In standard CCS semantics instead, since the process knows the choice it has made, then two different reductions are possible, namely $a.\sigma + a.\tau \xrightarrow{a} \sigma$ and $a.\sigma + a.\tau \xrightarrow{a} \tau$, each corresponding to a different choice.

(3) At the containment relation level, we adopt a testing approach [De Nicola and Hennessy 1984; Hennessy 1988]. As a consequence, two processes are related by the subcontract relation $\preceq$ if all the clients compliant with the smaller process are also compliant with the larger one. Contrariwise, the conformance relation in [Fournet et al. 2004] is defined as the largest simulation relation that is consistent with stuck-freedom. As a result, the conformance relation of Fournet et al. is not complete with respect to stuck-freedom, in the sense that there are processes that are stuck-free exactly in the same contexts but are not related by conformance. For example, $a.(b \oplus c)$ and $a.b + a.c$ are stuck-free equivalent but are not conformance equivalent: since they are not bisimilar, then in the setting of Fournet et al.—which is a "subjective" one—it would be wrong to replace one for the other. In our context instead we want to be able to replace services as long as their *external* observable behaviours are compatible, which is why in our theory the two processes above are equivalent. For the same reason, while in [Fournet et al. 2004] completeness with respect to stuck-freedom is not sought and conformance does not allow either width or depth subtyping, in our setting completeness is a key property since it ensures maximal substitutability for equivalent external behaviour and width or depth subtyping are key mechanisms for a modular application of substitutability.

This change of perspective—objective external observer vs. subjective internal observer—explains why, *mutatis mutandis* (*cf.* actions for successful termination, parallel composition, restrictions, etc.), one can roughly see our work as a complete characterization of stuck-freedom and, reciprocally, consider conformance as a stricter subcontracting relation that takes into account subjective aspects such as internal implementation.

Bravetti and Zavattaro [2007] propose a contract language equipped with a refinement relation. The language includes all the classical operators found in common process algebras (parallel composition, restriction, etc.), but is constrained so that output actions occur can only occur in the context of an internal choice. This restriction somehow resembles the design choice of session types and, not surprisingly, the refinement relation for this language allows width extensions without any intervening filtering. However, while the presence of parallel composition in the contract language suffices for modelling processes that offer output messages in such a way that the interacting party can externally choose the order in which they are consumed, the modelling of timeouts (*cf.* §5.3) would require an extension to the language of Bravetti and Zavattaro. We remark two further differences regarding the refinement relation in [Bravetti and Zavattaro 2007] and our subcontract relation: the first is that the refinement relation depends on (is indexed by) a set of (output) actions that may occur in the environment, and width subtyping is determined by this dependency in the sense that it is possible to extend behaviors with additional, input-guarded branches if one knows that no output message will

ever be able to trigger such branches. This roughly corresponds to a static form of action filtering in our setting and in that respect they adopt a solution similar to the one proposed in the work by Laneve and Padovani [2007] we discuss at the end of this section. Moreover, the refinement relation is determined in a symmetric way for all the participants of a system, whereas our notion of compliance is asymmetric (in favor of the client). This makes their refinement relation more demanding than ours. In particular, all the participants must successfully terminate, meaning that depth extensions are not entailed by refinement.

A very preliminary version of this work was presented at PLAN-X 2007 workshop [Castagna et al. 2007] and largely improved in the version presented one year later at POPL '08 [Castagna et al. 2008]. Although the PLAN-X workshop has just informal proceedings, these are available on the web. Therefore it seems worth discussing the differences of the present article both with the PLAN-X version and with the improved POPL version. While the overall presentation and structure of the three papers is the same, both this and the POPL versions improve over the PLAN-X one in several points. Here and in [Castagna et al. 2008] we consider a slightly different version of strong compliance relation which now coincides with the must testing preorder, while in PLAN-X strong compliance differed from must testing for some (uninteresting) pathological cases that involved the empty contract. The deduction system of PLAN-X was reworked in favor of elegance and simplicity. The resulting algebraic theory of filters is also cleaner. We present better results for language neutrality. Finally, the study of the algorithmic version of the deduction system, of its logical interpretation, and of the decidability of the containment relation, was absent from the PLAN-X version and introduced in the POPL one. The article presented here improves the work in POPL in several regards. Foremost, while in the work presented at POPL contracts (and filters) were finite, here the theory is defined for recursive contracts (and filters) by working directly with infinite recursive trees and by proposing two different finite representations for them (we believe that the in-depth treatment of infinite terms and of the relation with their finite representations constitutes a nice contribution of our work). This implied a complete reworking of most of the definitions and of the proofs (even though the latter were not included in the POPL proceedings for space reasons). The finite representations we introduce here are then used to study WSCL and WS-BPEL and possible implementations of filters are explored; in particular we outline how our theory can be used and implemented in the current specification of WS-BPEL without requiring any modification to the language or to existing WS-BPEL processes. All these practical aspects are completely absent in the work presented at POPL. Finally, the deduction system for filters is here further improved and we also use a different and (we hope) more elegant syntax for filters, by relying only on the underlying algebraic operators.

Starting from the PLAN-X work the third author and Cosimo Laneve proposed a simplification where contracts are "statically" filtered [Laneve and Padovani 2007]: each contract is associated with a *static interface* (in the sense that it does not change over time) declaring the only visible actions of the contract and blocking all the other ones whenever they happen. As stated in [Laneve and Padovani 2007], the resulting approach is less general than ours and, consequently, yields a stricter

subcontract relation. For instance, the relation $a.b \preceq (a.(a + b)) + b.c$, which we commented on just before §1.1, does not hold in the interface approach (for a practical example of relation that does not hold for interfaces see the contracts σ_2 and σ_2' in §2.2.2 and the explanation given at the end of §3.2.1). On the other hand, interfaces allow for simpler algorithmic treatment and implementation.

More recently Padovani [2008] has extended the framework in the present work by equipping filters with finite *buffers*. This extension allows filters to intercept messages exchanged between client and service, to temporarily save them in internal buffers, and to deliver them at a later stage, when client and service actually need them. As a consequence, the weak subcontract relation identifies larger classes of contracts. For example, it becomes possible to prove $a.b.\sigma \preceq b.a.\sigma$ as well as $\overline{a}.\overline{b}.\sigma \preceq \overline{b}.\overline{a}.\sigma$. More generally, input actions can be *delayed* and output actions can be *anticipated* ($a.\overline{b}.\sigma \preceq \overline{b}.a.\sigma$) but the converse is not true: $\overline{b}.a.\sigma \not\preceq a.\overline{b}.\sigma$. Intuitively this is because the client of $\overline{b}.a.\sigma$ may need the information contained in the received message $\overline{b}$ before it sends the message $\overline{a}$ back to the service. The extension of filters with buffering also allows one to enlarge the class of filters that can be efficiently and effectively implemented as pure CCS processes (§5.4).

7. CONCLUSION AND FUTURE WORK

This paper provides a foundation for behavioral typing of Web services by means of *contracts* and it promotes service reuse and/or redefinition by the introduction of a *subcontract relation*.

Our contract language is the sequential fragment of CCS without parallelism, without explicit internal moves, without relabelling, and without restriction [De Nicola and Hennessy 1987]. The fact that we focus on this simple language may at first appear overly restrictive, especially because of the lack of parallel composition. However, recall that we are only interested in describing the external, observable behavior of Web services, not their internal implementation. So, while it is reasonable to expect that a Web service is *internally* implemented with parallel processes and private communications, we take the point of view that the contract language should be kept as simple as possible, as long as it can faithfully describe the service behavior. As we have pointed out in the related work section, our contract language is just a concrete representation for Hennessy's acceptance trees [Hennessy 1985]. Well-known works [Hennessy 1988] show that—possibly infinite, but not necessarily regular—acceptance trees are a fully abstract model for fully-fledged variants of CCS. So, the only real restriction imposed by our contract language is regularity, which basically "limits" the application of the contract language to finite-state processes. In this respect, we observe first of all that working with finite-state processes/contracts clearly makes the whole theory decidable, and thus practically relevant. Second, it is a fact that concrete languages for Web service description and implementation (including WSDL, WSCL, and WS-BPEL itself) only deal with finite-state processes, at least as far as the observable, external behavior is concerned.

As regards the subcontract relation, we reconcile 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 behavior of clients. In summary, we tackle and solve the lack of transitivity that arises when combining reduction and extension refinements as defined by Derrick et al. [1996].

Filters provide the technical device that makes it possible. Although we initially defined filters essentially as technical mechanism for coupling 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 axiomatization 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 characterized 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 algebras, functions, ...). By defining some minimal requirements on the language (in a nut-shell, the observable behavior 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. We have also shown that we do not need either to extend WS-BPEL syntax or to reprogram existing WS-BPEL processes in order to apply to them the theory presented in this paper and thus reuse them in more contexts.

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

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

That is, a restriction is nothing but a recursive filter that allows all actions apart from the restricted one.

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 be answered with several different contracts requiring filters that may be unrelated 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 behaviors of its "canonical" service are considered mandatory and not to be filtered out. For instance, when searching for services implementing the behavior described in Figure 1 we could specify, along with the query, the filter $\mathtt{Login}.\overline{\mathtt{ValidLogin}}.\mathtt{Query}.\overline{\mathtt{Catalog}}.\mathtt{AddToCart}.\ \mathtt{Buy}.(\mathtt{CreditCard}.\overline{\mathtt{Valid}} \vee \mathtt{BankTransfer}.\overline{\mathtt{Valid}})$ thus obtaining only services that may complete a sale, avoid-

ing useless services such as those with contract $\texttt{Login}.\overline{\texttt{InvalidLogin}}$. This use of filters is similar to that proposed in [Bernardi et al. 2008] (*cf.* §6) where filters also define "roles" of a choreography, that is specifications of the minimal behaviour each component of the choreography must provide.

From a technical viewpoint this work introduces some novelties. An original aspect of this line of research is the use of a non-standard labeled semantics for CCS terms—which we first introduced in [Carpineti et al. 2006]—that captures the evolution of a process from the perspective of an external observer rather than, as for standard CCS semantics, from the perspective of the process itself. We believe this to be the right perspective for analyzing observable behaviour of web-services even though, as a matter of fact, this feature is not strictly necessary (one could equivalently use standard CCS semantics as done by Laneve and Padovani [2007], Padovani [2008], and Bernardi et al. [2008]). Another technical novelty introduced here is the use of infinite trees to represent recursive terms: although this technique is recurrently used in functional programming context, in our ken this is the first usage in the setting of concurrency theory where systems always rely upon some particular concrete syntax for recursive definitions. Similarly, while in a functional setting coercions and their interpretation via Curry-Howard isomorphism as proofs of some containment relation are not surprising, we never met them before in the realm of concurrency.

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

—*Higher-order contracts:* In the current formalism synchronization does not carry any information. Thus a natural next 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 synchronization with a). It is unclear to which extent such constructs would affect the $\preceq$ preorder over contracts.

—*Contract morphisms:* 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 or reordering of actions) and to perform service discovery modulo such morphisms: when searching for services of a given contract a client could be returned a service and a conversion function that adapts the interaction pattern of the client to the service at issue (somewhat similar to, but less stringent than, libraries searches modulo type isomorphisms [Rittri 1993; Di Cosmo 1995]). This could set the basis of a new theory of orchestration where light and highly distributed orchestrators would be implemented by filters and contract morphisms.

As explained in the section on related work (§6), a step in this direction has already been done in [Padovani 2008]. This approach 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 of the work on semantic subtyping [Castagna and Frisch 2005; Frisch et al. 2008].

—*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 realizability semantics for contracts is worth exploring.

REFERENCES

Alves, A., Arkin, A., Askary, S., Barreto, C., et al. 2007. *Web Services Business Process Execution Language Version 2.0.* OASIS Standard, `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`.

Baeten, J. C. M., Corradini, F., and Grabmayer, C. A. 2007. A characterization of regular expressions under bisimulation. *J. ACM 54,* 2, 6.

Banerji, A., Bartolini, C., Beringer, D., Chopella, V., et al. 2002. *Web Services Conversation Language (*wscl*) 1.0.* W3C Note, `http://www.w3.org/TR/2002/NOTE-wscl10-20020314`.

Bellwood, T., Capell, S., Clement, L., Colgrave, J., et al. 2005. uddi *Version 3.0.2.* OASIS Standard, `http://uddi.org/pubs/uddi-v3.0.2-20041019.htm`.

Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., and Mecella, M. 2003. Automatic composition of e-services that export their behavior. In *In Proc. 1st Int. Conf. on Service Oriented Computing (ICSOC), volume 2910 of LNCS*. Springer, 43–58.

Bernardi, G., Bugliesi, M., Macedonio, D., and Rossi, S. 2008. A theory of adaptable contract-based service composition. In *Proc. of Workshop on Global Computing Models and Technologies (GlobalComp'08)*. IEEE Computer Society Press. To appear.

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

Brinksma, E., Scollo, G., and Steenbergen, C. 1995. Lotos specifications, their implementations and their tests. 468–479.

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

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

Carbone, M., Honda, K., and Yoshida, N. 2007b. Structured communication-centred programming for web services. In *ESOP '07, 16th European Symposium on Programming*. LNCS 4421. Springer.

Cardelli, L. 1988. A semantics of multiple inheritance. *Information and Computation 76*, 138–164.

Carpineti, S., Castagna, G., Laneve, C., and Padovani, L. 2006. A formal account of contracts for Web Services. In *3rd Int. Workshop on Web Services and Formal Methods*. LNCS 4184. Springer.

Castagna, G., Dezani-Ciancaglini, M., Giachino, E., and Padovani, L. 2008. General session types. Technical Report id.: hal-00334435, CNRS - PPS, University Paris 7. `http://hal.archives-ouvertes.fr/hal-00334435`.

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

Castagna, G., Gesbert, N., and Padovani, L. 2007. A theory of contracts for web services. In *PLAN-X '07, 5th ACM-SIGPLAN Workshop on Programming Language Technologies for XML*.

Castagna, G., Gesbert, N., and Padovani, L. 2008. A theory of contracts for web services. In *POPL '08, 35th ACM Symposium on Principles of Programming Languages*. 261–272.

Chen, G. 2004. Soundness of coercion in the calculus of constructions. *Journal of Logic and Computation 14,* 3, 405–427.

Chinnici, R., Haas, H., Lewis, A.-A., Moreau, J.-J., Orchard, D., and Weerawarana, S. 2007. *Web Services Description Language (*wsdl*) Version 2.0 Part 2: Adjuncts.* W3C Recommendation, `http://www.w3.org/TR/wsdl20-adjuncts/`.

Chinnici, R., Moreau, J.-J., Ryman, A., and Weerawarana, S. 2007. *Web Services Description Language (*wsdl*) Version 2.0 Part 1: Core Language.* W3C Recommendation, `http://www.w3.org/TR/wsdl20/`.

Courcelle, B. 1983. Fundamental properties of infinite trees. *Theoretical Computer Science 25*, 95–169.

De Giacomo, G. and Sardiña, S. 2007. Automatic synthesis of new behaviors from a library of available behaviors. In *IJCAI*. 1866–1871.

De Nicola, R. and Hennessy, M. 1984. Testing equivalences for processes. *Theoretical Computer Science 34*, 83–133.

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

De Nicola, R. and Labella, A. 2003. Nondeterministic regular expressions as solutions of equational systems. *Theor. Comput. Sci. 302,* 1-3, 179–189.

Derrick, J., Bowman, H., Boiten, E., and Steen, M. 1996. Comparing LOTOS and Z refinement relations. In *FORTE/PSTV'96*. Chapman & Hall, Kaiserslautern, Germany, 501–516.

Di Cosmo, R. 1995. *Isomorphisms of Types: from Lambda Calculus to Information Retrieval and Language Design*. Birkhäuser.

Fallside, D. C. and Walmsley, P. 2004. *XML Schema Part 0: Primer Second Edition*. W3C Recommendation, `http://www.w3.org/TR/xmlschema-0/`.

Fournet, C. and Gonthier, G. 1996. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of the 23rd ACM Symposium on Principles of Programming Languages*. ACM, St. Petersburg Beach, Florida, 372–385.

Fournet, C., Gonthier, G., Lévy, J.-J., Maranget, L., and Rémy, D. 1996. A calculus of mobile agents. In *CONCUR*. Lecture Notes in Computer Science, vol. 1119. Springer, 406–421.

Fournet, C., Hoare, C. A. R., Rajamani, S. K., and Rehof, J. 2004. Stuck-free conformance. In *CAV'04*. LNCS 3114. Springer.

Frisch, A., Castagna, G., and Benzaken, V. 2008. Semantic subtyping: dealing set-theoretically with function, union, intersection, and negation types. *Journal of the ACM 55,* 4, 1–64.

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

Hennessy, M. 1985. Acceptance trees. *Journal of the ACM 32,* 4, 896–928.

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

Honda, K. 1993. Types for dyadic interaction. In *CONCUR '93*. LNCS 715. Springer, 509–523.

Honda, K., Vasconcelos, V. T., and Kubo, M. 1998. Language primitives and type discipline for structured communication-based programming. In *European Symposium on Programming*. LNCS 1381. Springer.

Hull, R., Benedikt, M., Christophides, V., and Su, J. 2003. E-services: a look behind the curtain. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, New York, NY, USA, 1–14.

Laneve, C. and Padovani, L. 2006. Smooth orchestrators. In *FoSSaCS* (2006-04-05), L. Aceto and A. Ingólfsdóttir, Eds. LNCS, vol. 3921. Springer, 32–46.

Laneve, C. and Padovani, L. 2007. The *must* preorder revisited – An algebraic theory for web services contracts. In *18th International Conference on Concurrency Theory*. LNCS 4703, Springer.

Laneve, C. and Padovani, L. 2008. The pairing of contracts and session types. LNCS, vol. 5065. Springer, 681–700.

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

OCaml. Objective Caml. `http://caml.inria.fr/ocaml/`.

Padovani, L. 2008. Contract-directed synthesis of simple orchestrators. LNCS, vol. 5201. Springer, 131–146.

Pistore, M., Traverso, P., Bertoli, P., and Marconi, A. 2005. Automated synthesis of composite bpel4ws web services. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services*. IEEE Computer Society, Washington, DC, USA, 293–301.

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

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

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

Traverso, P. and Pistore, M. 2004. Automated composition of semantic web services into executable processes. In *International Semantic Web Conference*. 380–394.