

Contract-Based Discovery and Composition of Web Services*

Mario Bravetti Gianluigi Zavattaro

Department of Computer Science, University of Bologna, Italy

Abstract. In the context of Service Oriented Computing behavioural contracts are descriptions of the observable message-passing behaviour of services. In other terms, contracts are behavioural interfaces that can be used, for instance, to check whether a group of services can be safely combined avoiding, e.g., undesired deadlocks. In this paper we consider the problem of discovering available services that can be used to implement a given service system. The idea is to first design a service system by describing the overall behaviour of each of its participant, and then instantiate such participants retrieving services exposing a behavioural contract which is conformant with the corresponding given behaviour.

1 Introduction

Service Oriented Computing (SOC) is a paradigm for distributed computing based on services intended as autonomous and heterogeneous components that can be published and discovered via standard interface languages and publish/discovery protocols. Web Services are the most prominent service oriented technology: Web Services publish their interface expressed in WSDL, they are discovered through the UDDI protocol, and they are invoked using SOAP.

Even if one of the declared goal of Web Services is to support the automatic discovery of services, this is not yet practically achieved. Two main problems are still to be satisfactorily solved. The first one, investigated by the semantic web research community, is concerned with the lack of semantic information in the description of services. The second problem, addressed in this paper, is concerned with the problem of guaranteeing that the interacting services are compliant in the sense that their behaviours are complementary. In particular, it is important to check whether in a set of services, combined in order to collaborate, no service deadlocks waiting indefinitely for a message that never arrives.

In order to be able to check the compliance of the composed services, it is necessary that the services expose in their interface also the description of their expected behaviour. In the service oriented computing literature, this kind of information is referred to as the behavioural *service contract* [15]. More precisely, the service contract describes the sequence of input/output operations that the service intends to execute within a session of interaction with other services.

* Research partially funded by EU Integrated Project Sensoria, contract n. 016004.

Compliance checking based on the behavioural descriptions of the composed entities has been already considered, for instance, in the context of component-based systems (see e.g. [9, 2, 23]) or for client-service interaction [14]. In this paper, we consider a different scenario with respect to both approaches.

As far as component-based systems are concerned, the commonly adopted approach is to synthesize either *wrappers* or *adaptors* that respectively block (the non compatible) part of the behaviour of one component or deal with possible mismatches between the combined components. The approach adopted in this paper is different because we address the problem of composition without the introduction of any additional wrapper or adaptor. In other terms, we consider the problem of retrieving some already available services in order to implement a correct composition without the introduction of any additional element. In the service oriented computing literature, the approach we consider is known with the name of *choreography* [27], which contrasts with the *orchestrated* approach [25] according to which all services communicate only with a central orchestrator. It is worth mentioning the fact that we could define our theory having in mind components instead of services. Nevertheless, our assumption about the choreographic approach makes all our theory more related to the current vision of service oriented computing.

As far as client-service interaction is concerned, we assume a more general context in which an arbitrary number of interacting services communicate directly without the presence of any central coordinator. We call this different context *multi-party* composition. Moving from a simpler client-service to a more complex multi-party scenario introduces several interesting new problems such as independent refinement. By independent refinement we mean the possibility to replace several services in a composition with other services that are selected one independently from the other ones.

More precisely, the aim of this paper is to exploit the notion of behavioural service contracts in order to define a theory that, on the one hand, permits to formally verify whether they are compliant (thus giving rise to a correct composition) and, on the other hand, permits to replace a service with another one without affecting the correctness of the overall system. In this case we say that the initially expected contract is replaced with one of its *subcontracts*.

We intend to formalize a notion of subcontract to be exploited in the *service discovery* phase. Consider, for instance, a service system defined in terms of the behavioural contracts to be fulfilled by each of the service components. The actual services to be combined could be retrieved independently one from the other (e.g. querying contemporaneously different service registries) collecting services that either expose the expected contract, or one of its subcontracts. Another application that we foresee for our notion of subcontract is for *service updates*, as a mean to ensure backward compatibility. Consider, e.g., a service that should be updated in order to provide new functionalities; if the new version exposes a subcontract of the previous service, our theory ensures that the new service is a correct substitute for the previous one.

1.1 Technical contribution

The main contribution of this paper is to generalize results that we have presented in [3, 7], where we have presented for the first time our approach for the definition of a subcontract relation: we first assume that this relation should be a pre-order, then we formalize the property that we want a “good” subcontract pre-order should preserve, and finally we define our relation as the maximum of such pre-orders (i.e. the union of all pre-orders satisfying the considered property). This is similar to the co-inductive approach considered, e.g., in the definition of the bisimulation relation for CCS [24]. More precisely, in [3, 7] we have presented a theory, developed following this approach, considering two specific languages, one for contracts and one for service systems. In this paper we generalize such theory in two ways. On the one hand, we do not consider any specific contract language thus presenting a version of our theory that can be applied to any contract language satisfying a property, called *output persistence*, that we will discuss in the following. On the other hand, we do not make any specific assumption on the way service systems are specified (in [3, 7] we defined a subcontract relation assuming a precise form of service system specifications in which the restriction operator is applied directly to contracts and not to parallel compositions of contracts).

More formally, we consider a generic language for behavioural contracts essentially consisting of a process algebraic representation of labeled transition systems defined on internal, input and output actions, and an additional action representing successful completion. Then, we define a language for service system specification that simply allows for the composition of contracts with the parallel and restriction operators. We use this latter language to formalize the notion of compliance: n services/contracts are compliant if their composition is guaranteed to successfully complete without deadlocks or livelocks. After having formalized compliance, we are able to formalize the property that each refinement should satisfy: a refinement is a subcontract pre-order if it preserves service compliance, namely, given n compliant services, and substituting each of them with one of its refinements, the achieved n services are still compliant. Then we define the *subcontract relation* as the union of all subcontract pre-orders. One of the main results proved in this paper is that for the class of behavioural contracts that we consider, the subcontract relation achieved according to this approach is actually the largest subcontract pre-order thus allowing for the independent replacement/retrieval of contracts. In fact, in other theories of contracts recently proposed in the literature (details are reported in the next subsection), independent replacement is not allowed.

This difference with respect to other contract theories relies on the *output persistence* property that we impose on behavioural contracts: a contract is output persistent if once a contract reaches a state in which it can perform an output operation, this operation must be eventually executed from the contract before successful completion. This property is usually satisfied by languages for composing services, such as WS-BPEL [25], in which output operations cannot be guard

in external choices. In these languages, once an output action is executable by a process, this output must be executed before the process successfully completes.

Another important technical achievement of [3, 7] that we report in this paper is a characterization of the subcontract relation in a testing-like scenario [18]: we can prove that a contract C' is a subcontract of C if, after some appropriate transformations applied to both C' and C , the former is guaranteed to satisfy at least all the tests satisfied by the latter. In particular, we show how to use the theory of *should-testing* [26] to prove that one contract is a subcontract of another one. An important consequence of this characterization is a precise localization of our refinement with respect to traditional refinements such as failure refinement, or simulation (i.e. half-bisimulation): the refinement that we achieve as the largest one preserving compliance is coarser than both failure refinement and simulation.

1.2 Related work

As stated above, we resort to the theory of testing, in particular, to the must-testing pre-order. There are some significant differences between our form of testing and the traditional one proposed by De Nicola-Hennessy [18]. The most significant difference is that, besides requiring the success of the test, we impose also that the tested process should successfully complete its execution. This further requirement has important consequences; for instance, we do not distinguish between the always unsuccessful process $\mathbf{0}$ and other processes, such as $a.\mathbf{1} + a.b.\mathbf{1}$,¹ for which there are no guarantees of successful completion in any possible context. Another significant difference is in the treatment of divergence: we do not follow the traditional catastrophic approach, but the fair approach introduced by the theory of should-testing of Rensink-Vogler [26]. In fact, we do not impose that all computations must succeed, but that all computations can always be extended in order to reach success.

It is well known that the De Nicola-Hennessy must testing pre-order and the CSP failure refinement [20] coincide (at least for finitely branching processes without divergences [17]). It is interesting to say that the failure refinement has been already exploited for checking component compatibility by Allen and Garlan in [1]. Similarly to our theory, the failure refinement is used to prove that a component can be replaced by one of its refinements in a component composition. Differently from our theory, a composition of several components is obtained adding a *Glue* component which behaves as a mediator for every component interaction. This *Glue* component permits to cut the additional actions that the refined components may include. The main difference with our theory is that, in our context, we have no mediator that allows us to cut additional behaviours of refined services. Nevertheless, the output persistence property that we consider allows us to replace a service with another one having additional behaviour.

¹ We use $\mathbf{0}$ to denote unsuccessful termination, $\mathbf{1}$ for successful completion and $_ + _$ for choice composition.

Behavioural contracts have been initially introduced in the context of process calculi by Fournet et al. [19]. As far as service oriented computing is concerned, an initial theory of contracts for client-service interaction has been proposed by Carpineti et al. [14] and then independently extended along different directions by Bravetti and Zavattaro (see e.g. [3–5]) by Laneve and Padovani [22], and by Castagna et al. [16]

In [19] contracts are CCS-like processes; a generic process P is defined as compliant to a contract C if, for every tuple of names \tilde{a} and process Q , whenever $(\nu\tilde{a})(C|Q)$ is stuck-free then also $(\nu\tilde{a})(P|Q)$ is. Our notion of contract refinement differs from stuck-free conformance mainly because we consider a different notion of stuck process state. In [19] a process state is stuck (on a tuple of channel names \tilde{a}) if it has no internal moves (but it can execute at least one action on one of the channels in \tilde{a}). In our approach, an end-state different from successful termination is stuck (independently of any tuple \tilde{a}). Thus, we distinguish between internal deadlock and successful completion while this is not the case in [19]. Another difference follows from the exploitation of the restriction $(\nu\tilde{a})$; this is used in [19] to explicitly indicate the local channels of communication used between the contract C and the process Q . In our context we can make a stronger *closed-world* assumption (corresponding to a restriction on all channel names) because service contracts do not describe the entire behaviour of a service, but the flow of execution of its operations inside one session of communication.

The closed-world assumption is considered also in [14] where, as in our case, a service oriented scenario is considered. In particular, in [14] a theory of contracts is defined for investigating the compatibility between one client and one service. Our paper considers multi-party composition where several services are composed in a peer-to-peer manner. Moreover, we impose service substitutability as a mandatory property for our notion of refinement; this does not hold in [14] where it is not in general possible to substitute a service exposing one contract with another one exposing a subcontract. Another significant difference is that the contracts in [14] comprise also external mixed choices that do not satisfy the output persistence property.

The preliminary versions of this paper [3, 7] introduces several interesting new aspects not considered in the initial approach of Carpineti et al. For instance, we consider also contracts with an infinite behaviour admitting the recursive operator in contracts, we consider multi-party compositions, and we present how to resort to the theory of testing pre-orders. Moreover, in another paper [4] we also investigate a new stronger notion of correctness for contract systems in which we assume that output operations cannot wait indefinitely. This problem naturally arises when also unlimited contract behaviours are permitted. For instance, the three contracts

$$\bar{a}.\bar{b} \quad a.recX.(\bar{c}.d.X) \quad recX.(c.\bar{d}.X) + b$$

($recX._$ denotes the classical recursive definition operator) are typically assumed to be compliant as their composition is stuck-free. Nevertheless, the second output of the first contract can wait indefinitely due to the possible unlimited interaction between the second and the third contract. In [4] we address this problem:

we propose a new stronger notion of compliance, called *strong compliance*, and we present a new theory of contracts which is consistent with strong compliance.

In [5] we discuss how contracts can be exploited in a more general theory for choreography conformance. Choreography languages, used to describe from a global point of view the peer-to-peer interactions among services in a composition, have been already investigated in a process algebraic setting by Busi et al. [10, 11] and by Carbone et al. [12]. The notion of choreography conformance is in general used to check whether a service can play a specific role within a given choreography. In [5] we present a basic choreography language, and we define conformance between that language and a contract language as follows: we check conformance by projecting a choreography on the considered role, and then exploiting contract refinement.

The work of Carpineti et al. [14] discussed above has been extended by (some of) the original authors in two ways, in [22] by explicitly associating to a contract the considered input/output alphabet, in [16] by associating to services a dynamic filter which eliminates from the service behaviour those interactions that are not admitted by the considered contract.

The explicit information about the input/output alphabet used in [22] allows the corresponding theory of contracts to be applied also to multi-party compositions. Nevertheless, the complete symmetry between inputs and outputs in the contract language considered in [22], does not permit to achieve one of the most interesting property we prove in this paper, that is, independent contract refinement. In fact, according to [22] the contracts in a multi-party composition cannot be independently refined, because if a refinement includes more inputs or outputs with respect to the corresponding contract, these additional names cannot be part of the input/output alphabets of other refinements. As the refinement cannot be applied independently the theory of contracts in [22] does not admit parallel discovery of services in multi-party service systems.

The dynamic filters of [16], on the contrary, allow for independent refinement, at the price of synthesizing a specific filter used to eliminate the additional behaviours of refinements. Even if very interesting from a theoretical point of view, the practical application of filters is not yet clear. In fact, it is not possible to assume the possibility to associate a filter to a remote service. This problem can be solved in client-service systems, assuming that a co-filter is applied to the local client, but it is not clear how to solve this problem in multi-party systems composed of services running on different hosts.

Finally, also the work based on types (e.g. that in [21] and [13]) gives rise to notions of refinement in terms of subtyping. For instance, the work in [21] allows subsystems like $a.(P|b.Q)$ to be replaced by $a.P|b.Q$ under the knowledge that the context is of the kind $\bar{a}.P'|\bar{b}.Q'$, while in the work of [13] a subterm can be replaced by another one where inputs can be syntactically added in external choices and outputs can be syntactically added in internal choices. The latter approach leads to a notion of refinement which is included in the one obtained in this paper. In our approach, however, features like input external choice extension and internal choice reduction are inferred and not taken by

syntactical definition. The former approach is incomparable because it deals with very special cases: in fact, following our approach we have that $a|b$ does not refine $a.b$ because the latter is compliant with the pair of contracts \bar{b} and $b.\bar{a}.\bar{b}$, while this is not the case for the former.

1.3 Structure of the paper.

In Section 2 we introduce our notation for behavioural contracts. In Section 3 we introduce the model for the representation of service systems in terms of contract compositions. Section 4 presents our theory for contract refinement. Finally, Section 5 reports some conclusive remarks.

This paper is a generalised version of [7] where we consider a language independent modeling of contracts (here a contract is any output persistent labeled transition system while in [7] a specific contract language is considered) and we do not impose any constraint on the way contracts can be composed (in [7] only service systems with restriction directly applied to contracts are considered).

2 Behavioural Contracts

Contracts are defined as transition systems labeled over internal, input, and output action names. We first define the class of labeled transition systems of interest for this paper.

Definition 1. *A finite connected labeled transition system (LTS) with termination transitions is a tuple $\mathcal{T} = (S, \mathcal{L}, \longrightarrow, s_h, s_0)$ where S is a finite set of states, L is a set of labels, the transition relation \longrightarrow is a finite subset of $(S - \{s_h\}) \times (\mathcal{L} \cup \{\surd\}) \times S$ such that $(s, \surd, s') \in \longrightarrow$ implies $s' = s_h$, $s_h \in S$ represents a halt state, $s_0 \in S$ represents the initial state, and it holds that every state in S is reachable (according to \longrightarrow) from s_0 .*

In a finite connected LTS with termination transitions we use \surd transitions (leading to the halt state s_h) to represent successful termination. On the contrary, if we get (via a transition different from \surd) into a state with no outgoing transitions (like, e.g., s_h) then we represent an internal failure or a deadlock.

We assume a denumerable set of action names \mathcal{N} , ranged over by a, b, c, \dots . We use $\tau \notin \mathcal{N}$ to denote an internal (unsynchronizable) computation. In contracts the possible transition labels are the typical internal τ action and the input/output actions a, \bar{a} .

Definition 2. *A contract is a finite connected LTS with termination transitions, that is a tuple $(S, \mathcal{L}, \longrightarrow, s_h, s_0)$, where $\mathcal{L} = \{a, \bar{a}, \tau \mid a \in \mathcal{N}\}$, i.e. labels are either a receive (input) on some operation $a \in \mathcal{N}$ or an invoke (output) directed to some operation $a \in \mathcal{N}$ at some location l .*

In the following we introduce a process algebraic representation for contracts by using a basic process algebra (a simple extension of basic CCS [24] with

successful termination) with prefixes over $\{a, \bar{a}, \tau \mid a \in \mathcal{N}\}$ and we show that from the LTS denoting a contract we can derive a process algebraic term whose behaviour is the same as that of the LTS. In the algebra syntax, we use $\mathbf{0}$ and $\mathbf{1}$ to denote unsuccessful and successful termination, respectively.

Definition 3. *We consider a denumerable set of contract variables Var ranged over by X, Y, \dots . The syntax of contracts is defined by the following grammar*

$$\begin{array}{l} C ::= \mathbf{0} \mid \mathbf{1} \mid \alpha.C \mid C+C \mid X \mid \text{rec}X.C \\ \alpha ::= \tau \mid a \mid \bar{a} \end{array}$$

where $\text{rec}X.$ is a binder for the process variable X . The set of the contracts C in which all process variables are bound, i.e. C is a closed term, is denoted by \mathcal{P}_{con} . In the following we will often omit trailing “ $\mathbf{1}$ ” when writing contracts.

The structured operational semantics of contracts is defined in terms of a transition system labeled by $\mathcal{L} = \{a, \bar{a}, \tau \mid a \in \mathcal{N}\}$ obtained by the rules in Table 1 (plus symmetric rule for choice), where we take λ to range over $\mathcal{L} \cup \{\sqrt{}\}$. In particular the semantics of a contract $C \in \mathcal{P}_{con}$ gives rise to a finite connected LTS with termination transitions $(S, \mathcal{L}, \longrightarrow, \mathbf{0}, C)$ where S is the set of states reachable from C and \longrightarrow includes only transitions between states of S . Note that the fact that such a LTS is finite (i.e. finite-state and finitely branching) is a well-known fact for basic CCS [24] (and obviously the additional presence of successful termination does not change this fact).

$$\begin{array}{c} \mathbf{1} \xrightarrow{\sqrt{}} \mathbf{0} \qquad \alpha.C \xrightarrow{\alpha} C \\ \hline \frac{C \xrightarrow{\lambda} C'}{C+D \xrightarrow{\lambda} C'} \qquad \frac{C\{\text{rec}X.C/X\} \xrightarrow{\lambda} C'}{\text{rec}X.C \xrightarrow{\lambda} C'} \end{array}$$

Table 1. Semantic rules for contracts (symmetric rules omitted).

In Appendix A we formalize the correspondence between contracts and terms of \mathcal{P}_{con} by showing how to obtain from a contract $\mathcal{T} = (S, \mathcal{L}, \longrightarrow, s_h, s_0)$ a corresponding $C \in \mathcal{P}_{con}$ such that there exists a (surjective) homomorphism from the operational semantics of C to \mathcal{T} itself. In the light of this correspondence result, we can safely consider the introduced process algebra to develop general theories for behavioural contracts.

In the remainder of the paper we use the following notations: $C \xrightarrow{\lambda}$ to mean that there exists C' such that $C \xrightarrow{\lambda} C'$ and, given a string of labels $w \in \mathcal{L}^*$, that is $w = \lambda_1 \lambda_2 \dots \lambda_{n-1} \lambda_n$ (possibly empty, i.e., $w = \varepsilon$), we use $C \xrightarrow{w} C'$ to denote the sequence of transitions $C \xrightarrow{\lambda_1} C_1 \xrightarrow{\lambda_2} \dots \xrightarrow{\lambda_{n-1}} C_{n-1} \xrightarrow{\lambda_n} C'$ (in case of $w = \varepsilon$ we have $C' = C$, i.e., $C \xrightarrow{\varepsilon} C$).

2.1 Output persistence

We now introduce a property for behavioural contracts that we call *output persistence*. We first present the formal definition.

Definition 4. (Output persistence) *Let $C \in \mathcal{P}_{con}$ be a contract. It is output persistent if given $C \xrightarrow{w} C'$ with $C' \xrightarrow{\bar{a}}$ then: $C' \not\checkmark$ and if $C' \xrightarrow{\alpha} C''$ with $\alpha \neq \bar{a}$ then also $C'' \xrightarrow{\bar{a}}$.*

The output persistence property states that once a contract decides to execute an output, its actual execution is mandatory in order to successfully complete the execution of the contract. This property typically hold in languages for the description of service behaviours or for service orchestrations (see e.g. WS-BPEL) in which output actions cannot be used as guards in external choices (see e.g. the `pick` operator of WS-BPEL which is an external choice guarded on input actions). In these languages, when a process instance or an internal thread decides to execute an output actions, it will have to complete such action before successful completion.

In the context of process algebra with parallel composition a syntactical characterization that guarantees output persistence can be found in [3]. The idea is to require that every output prefix (i.e. the term $\bar{a}.P$) is preceded by an internal τ prefix (i.e. the above term always occurs in the larger term $\tau.\bar{a}.P$).

As we also anticipated in the introduction, the actual impact of output persistence (in turn coming from the asymmetric treatment of inputs and outputs) in our theory is the existence of maximal independent refinement. This statement will be made precise by means of a counter-example that we are going to present after the Definition 9 –we postpone the presentation of this example because we first need to formalize contract compositions as well as the notion of correct composition– and by the results presented in Section 4.

We now formalize a direct consequence of output persistence that we will use in the following.

Proposition 1. *Let $C \in \mathcal{P}_{con}$ be an output persistent contract such that $C \xrightarrow{w} C' \xrightarrow{\bar{a}}$. If $C' \xrightarrow{w'} C''$ and $C'' \not\checkmark$ then the string w' must include \bar{a} .*

Proof. We proceed by contradiction assuming that the premise in the statement of the Proposition holds even for a sequence of actions w' that does not include \bar{a} . As $C' \xrightarrow{\bar{a}}$, due to output persistence, if w' does not include \bar{a} then also $C'' \xrightarrow{\bar{a}}$. Output persistence then guarantees that $C'' \not\checkmark$, thus contradicting the premise $C'' \not\checkmark$. \square

In the remainder of the paper we restrict to output persistent contracts, namely, \mathcal{P}_{con} denotes the set of output persistent contracts from now on. Note that, it is meaningful to do this because: any derivative of an output persistent contract is again an output persistent contract (as it can be immediately inferred from Definition 4) and homomorphism preserves output persistence (two homomorphic contracts are either both output persistent or no one is).

3 Service Systems

We now introduce the calculus for modeling systems of composed contracts. This is an extension of the previous calculus; the basic terms are contracts under execution denoted with $[C]$. Such a notation is inspired by process algebras with locations in which brackets “[”, “]” are used to denote a located process.

Besides the parallel composition operator \parallel , we consider also restriction \backslash in order to model the possibility to open local channels of interaction among contracts. The restriction operator that we consider is non-standard because it distinguishes between input and output operations. For instance, in the system $[C]\backslash\{a, \bar{b}\}$ we have that C cannot perform inputs on a and cannot perform outputs on b . This operator is useful for the modeling of private directed channels. For instance, if we want to model the fact that the service $[C_1]$ is the unique receptor on a particular channel a , we can simply restrict all the other services on action a (and $[C_1]$ on \bar{a}):

$$[C_1]\backslash\bar{a} \parallel [C_2]\backslash a \parallel [C_3]\backslash a \parallel \dots \parallel [C_n]\backslash a$$

As another example, consider a system composed of two contracts C_1 and C_2 such that channel a is used for communications from C_1 to C_2 and channel b is used for communications along the opposite directions. We can model such system as follows:

$$([C_1]\backslash\{a, \bar{b}\}) \parallel ([C_2]\backslash\{\bar{a}, b\})$$

As a final example of the flexibility of the restriction operator \backslash we consider the system

$$(([C] \parallel [C']) \backslash a) \parallel [D]$$

where we have that C , C' and D can execute input actions on the name a , but the inputs of C and C' cannot synchronize with output actions performed by D , while the inputs of D can synchronize with outputs performed by C and C' .

Definition 5. (Contract composition) *The syntax of contract compositions is defined by the following grammar*

$$P ::= [C] \mid P \parallel P \mid P \backslash L$$

where $C \in \mathcal{P}_{con}$ and $L \subseteq \{a, \bar{a} \mid a \in \mathcal{N}\}$.

We use $P, P', \dots, Q, Q', \dots$ to range over terms representing contract compositions, also called *systems*. The set of systems is denoted by \mathcal{P}_{sys} . Moreover, given a set of action names L we denote with \bar{L} the set of complementary actions: $\bar{L} = \{\bar{a} \mid a \in L\} \cup \{a \mid \bar{a} \in L\}$.

In the following we will sometimes omit parenthesis “[]” when writing contract compositions.

The operational semantics of systems is defined by the rules in Table 2 (plus the omitted symmetric rules).

Note that given $[C]\backslash L$, we obtain a transition system homomorphic to that of the contract $C\{\mathbf{0}/\alpha.D \mid \alpha \in L\}$ or, equivalently, to that of $[C\{\mathbf{0}/\alpha.D \mid \alpha \in L\}]$,

$$\begin{array}{c}
\frac{C \xrightarrow{\lambda} C'}{[C] \xrightarrow{\lambda} [C']} \qquad \frac{P \xrightarrow{\lambda} P' \quad \lambda \neq \surd}{P \parallel Q \xrightarrow{\lambda} P' \parallel Q} \\
\\
\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\bar{a}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'} \qquad \frac{P \xrightarrow{\surd} P' \quad Q \xrightarrow{\surd} Q'}{P \parallel Q \xrightarrow{\surd} P' \parallel Q'} \qquad \frac{P \xrightarrow{\lambda} P' \quad \lambda \notin L}{P \parallel L \xrightarrow{\lambda} P' \parallel L}
\end{array}$$

Table 2. Semantic rules for contract compositions (symmetric rules omitted).

where $C\{\mathbf{0}/\alpha.D \mid \alpha \in L\}$ represents the syntactical substitution of $\mathbf{0}$ for every occurrence of any subterm $\alpha.D$ such that $\alpha \in L$. In the light of this equivalence result, we will overload the restriction operator applying it also to contracts: $C \parallel L = C\{\mathbf{0}/\alpha.D \mid \alpha \in L\}$. It is immediate to verify (by just applying definition 4) that, for any set L , the restriction of an output persistent contract is again an output persistent contract.

We are now ready to define our notion of correct composition of contracts. Intuitively, a system composed of contracts is *correct* if all possible computations may guarantee completion; this means that the system is both deadlock and, under the fairness assumption², livelock free (there could be an infinite computation, but given any possible prefix of this infinite computation, it can be extended to reach a successfully completed computation). Note that our notion of correctness simply checks the compliance of the composed services without verifying whether the replies computed by the services actually corresponds to some desired functionalities. Henceforth, our notion of correct service composition should not be confused with the classical notion of program correctness.

Definition 6. (Correct contract composition) *A system P is a correct contract composition, denoted $P \downarrow$, if for every P' such that $P \xrightarrow{\tau}^* P'$ there exists P'' such that $P' \xrightarrow{\tau}^* P'' \xrightarrow{\surd}$.*

As examples of correct contract compositions, you can consider $C_1 \parallel C_2$ with

$$\begin{array}{ll}
C_1 = a + b & C_2 = \tau.\bar{a} + \tau.\bar{b} \\
C_1 = a.b & C_2 = \bar{a}.\bar{b} \\
C_1 = a + b + c & C_2 = \tau.\bar{a} + \tau.\bar{b} \\
C_1 = (a.b) + (b.a) & C_2 = \bar{a}.\bar{b} + \bar{b}.\bar{a} \\
C_1 = \text{rec}X.(a.\bar{b}.(X + \mathbf{1})) & C_2 = \text{rec}X.(\tau.\bar{a}.b.(X + \mathbf{1}))
\end{array}$$

As an example of contract composition which is not correct we can consider $[\bar{a}.b] \parallel [a]$ in which the first service deadlocks after executing the first output action (thus successful completion cannot be reached). Another interesting example is

² The notion of fairness that we consider is the following: when a state is traversed infinitely often each of its outgoing transitions is not discarded infinitely often.

$[recX.(\tau.\bar{a}.b.(X + \mathbf{1}))][a.recX(\tau.\bar{b}.a.(X + \mathbf{1}))]$, in which only an infinite computation (livelock) is executed (also in this case successful completion cannot be reached). We can also consider an additional example in which we combine both deadlock and livelock: $[recX.(\tau.\bar{a}.b.X + \tau.\bar{c}.d)][recX.(\tau.\bar{b}.a.X + c)]$.

The design choice of requiring livelock freedom under the fairness assumption is related to considering the following aspects.

- it is unsatisfactory to just require deadlock freedom (accepting too many systems as correct ones as, e.g., the system with livelock discussed above $[recX.(\tau.\bar{a}.b.(X + \mathbf{1}))][a.recX(\tau.\bar{b}.a.(X + \mathbf{1}))]$ as often multi-party conversations of services are executed inside sessions and it seems natural to ask for the ability of all the party involved to successfully terminate for the session to finish;
- it is too demanding to require livelock freedom without the fairness assumption (discarding too many systems that intuitively should be correct in the presence of infinite computations as, e.g., the last pair of contracts $C_1 C_2$ presented in the example above).

4 Service Discovery

In this Section we introduce our theory of contracts. The basic idea is to have a notion of refinement of contracts such that, given a system composed of the contracts C_1, \dots, C_n , we can replace each contract C_i by one of its refinements C'_i without breaking the correctness of the system.

This notion of refinement is useful when considering the problem of service discovery. Given the specification of a contract composition (composed of the so called “initial contracts”), the actual services to be composed are discovered independently sending queries to registries. It could be the case that services with a contract which exactly correspond to the “initial contracts” are not available; in this case, it is fundamental to accept also different contracts that could be replaced without affecting the overall correctness of the system.

One of the peculiarities of our theory of refinement, is that we consider the possibility of relying on some knowledge about the “initial contracts”, in particular, the input and output actions that occur in them. A very important consequence of this knowledge, is that we have the guarantee that a contract can be refined by another one that performs additional external input actions on names that do not occur in the initial contracts. For instance, the contract a can be refined by $a + b$ if we know that b is not among the possible outputs of the other initial contracts.

Some additional simple examples of refinement follow. Consider the correct system $C_1 \parallel C_2$ with

$$C_1 = a + b \quad C_2 = \tau.\bar{a} + \tau.\bar{b}$$

We can replace C_1 with $C'_1 = a + b + c$ or C_2 with $C'_2 = \bar{a}$ without breaking the correctness of the system. This example shows a first important intuition: a contract could be replaced with another one that has more external nondeterminism and/or less internal nondeterminism.

Consider now

$$D_1 = a + b + c \quad D_2 = \tau.\bar{a} + \tau.\bar{b}$$

where we can refine D_1 with $D'_1 = a + b + d$. Clearly, this refinement does not hold in general because we could have another correct system

$$D_1 = a + b + c \quad D'_2 = \tau.\bar{a} + \tau.\bar{b} + \tau.\bar{c}$$

where such a refinement does not hold. This second example shows that refinement is influenced by the potential actions that could be executed by the other contracts in the system. Indeed, D'_1 is not a correct substitute for D_1 because D'_2 has the possibility to produce \bar{c} .

Based on this intuition, we parameterize our notion of subcontract relation $C' \leq_{I,O} C$ on the set I of inputs, and the set O of outputs, that could be potentially executed by the other contracts in the system. We will see that $D'_1 \leq_{\mathcal{N}, \mathcal{N} - \{c,d\}} D_1$ but $D'_1 \not\leq_{\mathcal{N}, \mathcal{N}} D_1$.

4.1 Subcontract pre-orders as correctness preserving refinements

We first introduce the notion of context: a context is a system in which some contract is left unspecified and abstractly represented by a contract variable. In this way we can use contexts to represent architectures of systems in which we use contract variables to represent placeholders for contracts.

Definition 7. (Contexts) *Contexts \mathcal{C} are terms over the same syntax as contract compositions P with the addition of the term $[\mathcal{X}]$:*

$$P ::= [\mathcal{X}] \mid [C] \mid P \parallel P \mid P \setminus L$$

where \mathcal{X} is a contract variable belonging to a totally ordered set $CVar$, ranged over by $\mathcal{X}, \mathcal{Y}, \dots$. In the following we assume contexts \mathcal{C} not to include multiple occurrences of the same variable \mathcal{X} .

Given a set of terms W_i , with $1 \leq i \leq n$, where either $W_i \in \mathcal{P}_{con}$ (i.e. W_i is a contract) or $W_i \in CVar$ and a context \mathcal{C} such that n is the number of term variables included in \mathcal{C} , we use $\mathcal{C}(W_1, \dots, W_n)$ to stand for $\mathcal{C}\{W_i/\mathcal{X}_i \mid 1 \leq i \leq n\}$, where $\mathcal{X}_1, \dots, \mathcal{X}_n$ are the term variables included \mathcal{C} ordered according to the order on the set $CVar$.

As the notion of refinement that we define is parametrized on the sets of input and output actions that can be performed by the other contracts in the system, we need to formally define these sets.

Definition 8. (Input and Output sets) *Given the contract $C \in \mathcal{P}_{con}$, we define $I(C)$ (resp. $O(C)$) as the subset of \mathcal{N} of the potential input (resp. output) actions of C . Formally, we define $I(C)$ as follows:*

$$\begin{aligned} I(\mathbf{0}) &= I(\mathbf{1}) = I(X) = \emptyset & I(\tau.C) &= I(\bar{a}.C) = I(recX.C) = I(C) \\ I(a.C) &= \{a\} \cup I(C) & I(C+C') &= I(C) \cup I(C') \end{aligned}$$

and $O(C)$ as follows:

$$\begin{aligned} O(\mathbf{0}) = O(\mathbf{1}) = O(X) = \emptyset & & O(\tau.C) = O(a.C) = O(\text{rec}X.C) = O(C) \\ O(\bar{a}.C) = \{a\} \cup O(C) & & O(C+C') = O(C) \cup O(C') \end{aligned}$$

Given the system P , we define $I(P)$ (resp. $O(P)$) as the subset of \mathcal{N} of the potential input (resp. output) actions of P . Formally, we define $I(P)$ as follows:

$$I([C]) = I(C) \quad I(P\|P') = I(P) \cup I(P') \quad I(P\|L) = I(P) - \{a \mid a \in L\}$$

and $O(P)$ as follows:

$$O([C]) = O(C) \quad O(P\|P') = O(P) \cup O(P') \quad O(P\|L) = O(P) - \{a \mid \bar{a} \in L\}$$

Given a context C containing exactly one contract variable \mathcal{X} , we define $I(C)$ (resp. $O(C)$) as the subset of \mathcal{N} of the potential input (resp. output) actions of contracts in the context that can possibly synchronize with actions executed by any contract replacing \mathcal{X} in the context. Formally, assuming that C contains exactly one variable \mathcal{X} we define $I(C)$ as follows:

$$I([\mathcal{X}]) = \emptyset \quad I(C\|P) = I(P\|C) = I(C) \cup (I(P) - \text{outr}(C)) \quad I(C\|L) = I(C)$$

and

$$O([\mathcal{X}]) = \emptyset \quad O(C\|P) = O(P\|C) = O(C) \cup (O(P) - \text{inr}(C)) \quad O(C\|L) = O(C)$$

where $\text{inr}(C)$ (resp. $\text{outr}(C)$) are the input (resp. output) actions on which the variable \mathcal{X} occurring in C is restricted defined as follows:

$$\begin{aligned} \text{inr}([\mathcal{X}]) = \emptyset & & \text{inr}(C\|P) = \text{inr}(P\|C) = \text{inr}(C) \\ \text{inr}(C\|L) = \text{inr}(C) \cup \{a \mid a \in L\} & & \end{aligned}$$

and

$$\begin{aligned} \text{outr}([\mathcal{X}]) = \emptyset & & \text{outr}(C\|P) = \text{outr}(P\|C) = \text{outr}(C) \\ \text{outr}(C\|L) = \text{outr}(C) \cup \{a \mid \bar{a} \in L\} & & \end{aligned}$$

We are now ready to define the notion of subcontract pre-order $C'_i \leq_{I,O} C_i$ in which the substitutability of contract C_i with C'_i is parameterized in the possible input and output actions I and O of the other contracts in the considered system.

More precisely, we consider a correct system $\mathcal{C}(C_1, \dots, C_n)$, and we require that the system is still correct even if we replace each C_i with any C'_i such that $C'_i \leq_{I(C'), O(C')} C_i$ where $C' = \mathcal{C}(C_1, \dots, C_{i-1}, \mathcal{X}, C_{i+1}, \dots, C_n)$.

Definition 9. (Subcontract pre-order family) A family $\{\leq_{I,O} \mid I, O \subseteq \mathcal{N}\}$ of pre-orders over \mathcal{P}_{con} is a subcontract pre-order family if, for any context \mathcal{C} with n contract variables and including no contracts, contracts $C_1, \dots, C_n \in \mathcal{P}_{con}$ and $C'_1, \dots, C'_n \in \mathcal{P}_{con}$, and $\mathcal{C}_i = \mathcal{C}(C_1, \dots, C_{i-1}, \mathcal{X}, C_{i+1}, \dots, C_n)$, we have

$$\begin{aligned} \mathcal{C}(C_1, \dots, C_n) \downarrow \wedge \forall i. (C'_i \leq_{I_i, O_i} C_i \wedge I(\mathcal{C}_i) \subseteq I_i \wedge O(\mathcal{C}_i) \subseteq O_i) \\ \Rightarrow \mathcal{C}(C'_1, \dots, C'_n) \downarrow \end{aligned}$$

In the next subsection we will prove that there exists a maximal subcontract pre-order family; this is a direct consequence of the output persistence property. In fact, if we consider possible outputs that can disappear without being actually executed (as in an external choice among outputs $\bar{a} + \bar{b}$ or in a mixed choice $a + \bar{b}$ in which, e.g., the possible \bar{b} is no longer executable after the output or input on a) it is easy to prove that there exists no maximal subcontract pre-order family. Now consider, e.g., the trivially correct system $C_1 \parallel C_2$ with $C_1 = a$ and $C_2 = \bar{a}$; we could have two subcontract pre-order families \leq^1 and \leq^2 such that

$$a + c.\mathbf{0} \leq_{\mathcal{N}-c, \mathcal{N}-c}^1 a \quad \text{and} \quad \bar{a} + c.\mathbf{0} \leq_{\mathcal{N}-c, \mathcal{N}-c}^1 \bar{a}$$

and

$$a + \bar{c}.\mathbf{0} \leq_{\mathcal{N}-c, \mathcal{N}-c}^2 a \quad \text{and} \quad \bar{a} + \bar{c}.\mathbf{0} \leq_{\mathcal{N}-c, \mathcal{N}-c}^2 \bar{a}$$

but no subcontract pre-order family \leq could have

$$a + c.\mathbf{0} \leq_{\mathcal{N}-c, \mathcal{N}-c} a \quad \text{and} \quad \bar{a} + \bar{c}.\mathbf{0} \leq_{\mathcal{N}-c, \mathcal{N}-c} \bar{a}$$

because if we refine C_1 with $a + c.\mathbf{0}$ and C_2 with $\bar{a} + \bar{c}.\mathbf{0}$ we achieve the incorrect system $a + c.\mathbf{0} \parallel \bar{a} + \bar{c}.\mathbf{0}$ that can deadlock after synchronization on channel c . Note that, if we instead assume output persistence of contracts, as we do in this paper, subcontracts cannot add reachable outputs on new types. For instance an output persistent contract $a + \tau.\bar{c}$ adding a new output on c with respect to a , similarly to the pre-order \leq^2 in the example above, would not be a correct subcontract because when composed in parallel with the other initial contract \bar{a} would lead to a deadlock.

The existence of the maximal subcontract pre-order family permits to define co-inductively a subcontract relation achieved as union of all subcontract pre-orders. The co-inductive definition allows us to prove that two contracts are in subcontract relation, simply showing the existence of a subcontract pre-order which relates them. Moreover, we can use different subcontract pre-orders to refine independently several contracts in a multi-party composition, without affecting the correctness of the overall system.

4.2 Input-output subcontract relation as the maximal subcontract pre-order

We will show that (over output persistent contracts) the maximal subcontract pre-order family exists, and we will characterize it with a relation on contracts called the *input-output subcontract relation*. Differently from the subcontract pre-orders, that permit to refine contemporaneously several contracts in a composition, this new relation allows for the refinement of one contract only. Besides giving the possibility to prove the existence of the maximal subcontract pre-order family, this relation will allow us to resort to the theory of testing in the next subsection.

Before presenting the definition of the input-output subcontract relation, we present a coarser form of subcontract pre-order, called the *singular subcontract*

pre-order, according to which, given any system composed of a set of contracts, refinement is applied to one contract only (thus leaving the other unchanged). This new pre-order will allow us to prove that the input-output subcontract relation is coarser than any subcontract pre-order.

Definition 10. (Singular subcontract pre-order family) *A family $\{\leq_{I,O} \mid I, O \subseteq \mathcal{N}\}$ of pre-orders over \mathcal{P}_{con} is a singular subcontract pre-order family if, for any context \mathcal{C} with a single contract variable (and possibly some contracts) and $C, C' \in \mathcal{P}_{con}$ we have*

$$\mathcal{C}(C) \downarrow \wedge C' \leq_{I,O} C \wedge I(\mathcal{C}) \subseteq I \wedge O(\mathcal{C}) \subseteq O \quad \Rightarrow \quad \mathcal{C}(C') \downarrow$$

The following Proposition shows that a subcontract pre-order family is also a singular subcontract pre-order family. Intuitively, this means that if we can refine several contracts in a system without affecting its correctness, we can also refine only one of those contracts, leaving the others unchanged.

Proposition 2. *If a family of pre-orders $\{\leq_{I,O} \mid I, O \subseteq \mathcal{N}\}$ is a subcontract pre-order family then it is also a singular subcontract pre-order family.*

Proof. Suppose that $\{\leq_{I,O} \mid I, O \subseteq \mathcal{N}\}$ is a subcontract pre-order family. Consider a context \mathcal{C} with one contract variable \mathcal{X}_1 and $n-1$ contracts. Let $C \in \mathcal{P}_{con}$ be a contract such that $\mathcal{C}(C) \downarrow$. Consider now $C' \in \mathcal{P}_{con}$ such that $C' \leq_{I,O} C$ for I, O such that $I(\mathcal{C}) \subseteq I$ and $O(\mathcal{C}) \subseteq O$. We now prove that also $\mathcal{C}(C') \downarrow$. Let \mathcal{C}' be the context obtained from \mathcal{C} replacing all contracts C_2, \dots, C_n with the contract variables $\mathcal{X}_2, \dots, \mathcal{X}_n$. We have that $\mathcal{C}(C) = \mathcal{C}'(C, C_2, \dots, C_n)$. By $\mathcal{C}(C) \downarrow$ we have that also $\mathcal{C}'(C, C_2, \dots, C_n) \downarrow$. As $I(\mathcal{C}) = I(\mathcal{C}'(\mathcal{X}, C_2, \dots, C_n))$ and $O(\mathcal{C}) = O(\mathcal{C}'(\mathcal{X}, C_2, \dots, C_n))$ we can replace C with C' , and by reflexivity of pre-orders we can replace all other contracts C_i with themselves (in fact, we have that $D \leq_{I,O} D$ for every I, O and D). Thus, also $\mathcal{C}'(C', C_2, \dots, C_n) \downarrow$. The proof is completed by observing that $\mathcal{C}'(C', C_2, \dots, C_n) = \mathcal{C}(C')$ thus also $\mathcal{C}(C') \downarrow$. \square

We now have to prove that the singular subcontract pre-order families have maximum. This result is obtained in two steps: we first observe that we can restrict the set of contexts of interest in the definition of the singular subcontract pre-order families to contexts of the form $[\mathcal{X}] \parallel P$, then we define the maximum of the singular subcontract pre-order families considering this restricted set of contexts (this new relation is called *input-output subcontract relation*).

The first observation is formalized by the following Proposition.

Proposition 3. *Let \mathcal{C} be a context with a single contract variable. There exists $P \in \mathcal{P}_{sys}$ such that: $I(P) = I(\mathcal{C})$, $O(P) = O(\mathcal{C})$, and for every contract $C \in \mathcal{P}_{con}$ we have that $\mathcal{C}(C) \downarrow$ if and only if $C \parallel P \downarrow$.*

Proof. The proof is in Appendix B. \square

In the light of the above proposition we can conclude that we could restrict the contexts in the Definition 10 only to contexts of the form $[\mathcal{X}]\|P$. We now define the maximum of the singular subcontract pre-order families considering this restricted set of contexts. In this definition (and in the remainder of the paper) we use $\mathcal{P}_{sys,I,O}$ to denote the subset of processes of \mathcal{P}_{sys} such that $I(P) \subseteq I$ and $O(P) \subseteq O$.

Definition 11. (Input-Output Subcontract relation) *A contract C' is a subcontract of a contract C with respect to a set of input channel names $I \subseteq \mathcal{N}$ and output channel names $O \subseteq \mathcal{N}$, denoted $C' \preceq_{I,O} C$, if*

$$\forall P \in \mathcal{P}_{sys,I,O}. \quad (C\|P)\downarrow \Rightarrow (C'\|P)\downarrow$$

The main difference between the Definition 10 and the Definition 11 is that in the former we describe a property that every singular subcontract pre-order should satisfy, while in the latter we define a new relation (the input-output subcontract relation) that relates all those pairs of contracts C' and C that satisfy the same property. For instance, the identity relation is a singular subcontract pre-order because it satisfies the property, but it does not coincide with the input-output subcontract relation because it does not relate all those pairs of contracts C' and C that satisfy the property even if C' is not syntactically equal to C . In the following we will consider only the input-output subcontract relation (Definition 11), but we have presented both definitions because this simplifies the proof of the following Theorem (stating that the input-output subcontract relation includes all subcontract pre-order families) that, indeed, is a simple corollary of the Proposition 2 in which we made use of the Definition 10.

Theorem 1. *Given a subcontract pre-order family $\{\leq_{I,O} \mid I, O \subseteq \mathcal{N}\}$, we have that it is included in the family of pre-orders $\{\preceq_{I,O} \mid I, O \subseteq \mathcal{N}\}$, that is*

$$C' \leq_{I,O} C \Rightarrow C' \preceq_{I,O} C$$

Proof. From Proposition 2 we know that each subcontract pre-order family $\{\leq_{I,O} \mid I, O \subseteq \mathcal{N}\}$ is also a singular subcontract pre-order family. The thesis directly follows from the observation that the input-output subcontract relation $\preceq_{I,O}$ is the maximum of all singular subcontract pre-orders $\leq_{I,O}$, due to Proposition 3. \square

In the light of this last Theorem, the existence of the maximal subcontract pre-order family can be proved simply showing that $\{\preceq_{I,O} \mid I, O \subseteq \mathcal{N}\}$ is itself a subcontract pre-order family (thus it is the maximum among all subcontract pre-order families). The proof of this result (Theorem 2) is rather complex and requires several preliminary results.

The following proposition states an intuitive contravariant property: given $\preceq_{I',O'}$, and the greater sets I and O (i.e. $I' \subseteq I$ and $O' \subseteq O$) we obtain a smaller pre-order $\preceq_{I,O}$ (i.e. $\preceq_{I,O} \subseteq \preceq_{I',O'}$).

Proposition 4. Let $C, C' \in \mathcal{P}_{con}$ be two contracts, $I, I' \subseteq \mathcal{N}$ be two sets of input channel names and $O, O' \subseteq \mathcal{N}$ be two sets of output channel names. We have:

$$C' \preceq_{I,O} C \wedge I' \subseteq I \wedge O' \subseteq O \Rightarrow C' \preceq_{I',O'} C$$

Proof. The thesis follows from the fact that extending the sets of input and output actions means considering a greater set of discriminating contexts. \square

The following proposition states that a subcontract is still a subcontract even if we restrict its actions in order to consider only the inputs and outputs already available in the supercontract. The result about the possibility to restrict the outputs will be extensively used in the proof of Theorem 2.

Proposition 5. Let $C, C' \in \mathcal{P}_{con}$ be contracts and $I, O \subseteq \mathcal{N}$ be sets of input and output names. We have

$$\begin{aligned} C' \preceq_{I,O} C &\Rightarrow C' \setminus \overline{(O(C') - O(C))} \preceq_{I,O} C \\ C' \preceq_{I,O} C &\Rightarrow C' \setminus \overline{(O(C') - O(C))} \preceq_{I,O} C \end{aligned}$$

Proof. We discuss the result concerned with restriction of outputs (the proof for the restriction of inputs is symmetrical). Let $C' \preceq_{I,O} C$. Given any $P \in \mathcal{P}_{sys,I,O}$ such that $(C \parallel P) \downarrow$, we will show that $(C' \setminus \overline{(O(C') - O(C))} \parallel P) \downarrow$. We first observe that $(C \parallel P \setminus \overline{(O(C') - O(C))}) \downarrow$. Since $C' \preceq_{I,O} C$, we derive $(C' \parallel P \setminus \overline{(O(C') - O(C))}) \downarrow$. As a consequence $(C' \setminus \overline{(O(C') - O(C))} \parallel P \setminus \overline{(O(C') - O(C))}) \downarrow$. We can conclude $(C' \setminus \overline{(O(C') - O(C))} \parallel P) \downarrow$. \square

All the results discussed so far do not depend on the output persistence property. The first significant result depending on this peculiarity of the considered contracts is reported in the following proposition. It states that if we substitute a contract with one of its subcontract, the latter cannot activate outputs that were not included in the potential outputs of the supercontract.

Proposition 6. Let $C, C' \in \mathcal{P}_{con}$ be contracts and $I, O \subseteq \mathcal{N}$ be sets of input and output names. If $C' \preceq_{I,O} C$ we have that, for every $P \in \mathcal{P}_{sys,I,O}$ such that $(C \parallel P) \downarrow$,

$$(C' \parallel P) \xrightarrow{\tau}^* (C'_{der} \parallel P_{der}) \Rightarrow \forall a \in O(C') - O(C). C'_{der} \not\xrightarrow{\bar{a}}$$

Proof. We proceed by contradiction. Suppose that there exist C'_{der}, P_{der} such that $(C' \parallel P) \xrightarrow{\tau}^* (C'_{der} \parallel P_{der})$ and $C'_{der} \xrightarrow{\bar{a}}$ for some $a \in O(C') - O(C)$. We further suppose (without loss of generality) that such a path is minimal, i.e. no intermediate state $(C'_{der2} \parallel P_{der2})$ is traversed, such that $C'_{der2} \xrightarrow{\bar{a}}$ for some $a \in O(C') - O(C)$. This implies that the same path must be performable by $(C' \setminus \overline{(O(C') - O(C))} \parallel P)$, thus reaching the state $(C'_{der} \setminus \overline{(O(C') - O(C))} \parallel P_{der})$. However, since in the state C'_{der} of contract C' we have $C'_{der} \xrightarrow{\bar{a}}$ for some $a \in O(C') - O(C)$ and the execution of \bar{a} is disallowed by restriction, due to output persistence, the contract will never be able to reach success (no matter what contracts in P will do). Therefore $(C' \setminus \overline{(O(C') - O(C))} \parallel P) \not\downarrow$ and (due to Proposition 5) we reached a contradiction. \square

The following proposition permits to conclude that the set of potential inputs of the other contracts in the system is an information that does not influence the subcontract relation.

Proposition 7. *Let $C \in \mathcal{P}_{con}$ be contracts, $O \subseteq \mathcal{N}$ be a set of output names and $I, I' \subseteq \mathcal{N}$ be two sets of input names such that $O(C) \subseteq I, I'$. We have that for every contract $C' \in \mathcal{P}_{con}$,*

$$C' \preceq_{I,O} C \iff C' \preceq_{I',O} C$$

Proof. Let us suppose $C' \preceq_{I',O} C$ (the opposite direction is symmetric). Given any $P \in \mathcal{P}_{sys,I,O}$ such that $(C \parallel P) \downarrow$, we will show that $(C' \parallel P) \downarrow$. We first observe that $(C \parallel P \setminus (I - O(C))) \downarrow$. Since $C' \preceq_{I',O} C$ and $O(C) \subseteq I'$, we derive $(C' \parallel P \setminus (I - O(C))) \downarrow$. Due to Proposition 6 we have that $(C' \parallel P \setminus (I - O(C)))$ can never reach by τ transitions a state where outputs in $O(C') - O(C)$ are executable by some derivative of C' , so we conclude $(C' \parallel P) \downarrow$. \square

It is worth noting that a similar result does not hold for the output set, that is, if $O \subseteq O'$ we can have $C' \preceq_{I,O} C$ but $C' \not\preceq_{I,O'} C$ even under the assumption $I(C) \subseteq O, O'$. As an example, you can consider $\tau.\bar{a} + b \preceq_{\mathcal{N}, \mathcal{N}-b} \bar{a}$. This holds in general because the addition of an input on b is admitted in a subcontract if we know that no outputs on that channel can be executed by the other initial contracts in the system. On the contrary, we have that $\tau.\bar{a} + b \not\preceq_{\mathcal{N}, \mathcal{N}} \bar{a}$ because

$$[\bar{a}] \parallel [a.b] \parallel [\bar{b}]$$

is a correct composition while

$$[\tau.\bar{a} + b] \parallel [a.b] \parallel [\bar{b}]$$

is not, because the first and the second contracts are now in competition to consume the unique output on b produced by the third contract.

We are now in place to prove one of the main results of this paper, i.e., that the input-output subcontract relation defined in the Definition 11 is also a subcontract pre-order family.

Theorem 2. *The family of pre-orders $\{\preceq_{I,O} \mid I, O \subseteq \mathcal{N}\}$ is a subcontract pre-order family.*

Proof. Consider a context \mathcal{C} containing n contract variables and no contracts. Let $C_1, \dots, C_n \in \mathcal{P}_{con}$ be contracts such that $\mathcal{C}(C_1, \dots, C_n) \downarrow$. We consider contracts $C'_1, \dots, C'_n \in \mathcal{P}_{con}$ such that, for every i from 1 to n , $C'_i \preceq_{I_i, O_i} C_i$ for $I(C_i) \subseteq I_i$ and $O(C_i) \subseteq O_i$ with $\mathcal{C}_i = \mathcal{C}(C_1, \dots, C_{i-1}, \mathcal{X}, C_{i+1}, \dots, C_n)$.

We now derive that also $\mathcal{C}(C'_1, \dots, C'_n) \downarrow$.

We first observe that, for every i from 1 to n , we have that:

$$\mathcal{C}(C'_1 \setminus (\overline{O(C'_1)} - O(C_1)), \dots, C_i, \dots, C'_n \setminus (\overline{O(C'_n)} - O(C_n))) \downarrow$$

In fact, as the input-output subcontract relation is also a singular subcontract pre-order family, we can apply it $n - 1$ times to replace the contracts C_j with

$j \neq i$ with their subcontract $C_j \backslash \overline{(O(C'_j) - O(C_j))}$ (the latter is a subcontract of the former due to Proposition 5).

For any i , since $C'_i \preceq_{I_i, O_i} C_i$, by Proposition 6 we have that

$$\mathcal{C}(C'_1 \backslash \overline{(O(C'_1) - O(C_1))}, \dots, C'_i, \dots, C'_n \backslash \overline{(O(C'_n) - O(C_n))})$$

can never reach by τ transitions a state where outputs in $O(C'_i) - O(C_i)$ are executable by the derivative $C'_{i,der}$ of C'_i . If now we consider the behaviour of $\mathcal{C}(C'_1 \backslash \overline{(O(C'_1) - O(C_1))}, \dots, C'_n \backslash \overline{(O(C'_n) - O(C_n))})$ we derive that, for any i , we cannot reach by τ transitions a state

$$\mathcal{C}(C'_{1,der} \backslash \overline{(O(C'_1) - O(C_1))}, \dots, C'_{i,der} \backslash \overline{(O(C'_i) - O(C_i))}, \dots, \parallel C'_{n,der} \backslash \overline{(O(C'_n) - O(C_n))})$$

where $C'_{i,der}$ can execute outputs in $O(C'_i) - O(C_i)$. Hence the presence of the restriction operators does not affect the internal behaviour of

$$\mathcal{C}(C'_1 \backslash \overline{(O(C'_1) - O(C_1))}, \dots, C'_n \backslash \overline{(O(C'_n) - O(C_n))})$$

with respect to $\mathcal{C}(C'_1, \dots, C'_n)$. Therefore, we can finally derive $\mathcal{C}(C'_1, \dots, C'_n) \downarrow$ from

$$\mathcal{C}(C'_1 \backslash \overline{(O(C'_1) - O(C_1))}, \dots, C'_n \backslash \overline{(O(C'_n) - O(C_n))}) \downarrow$$

that is obtained by further applying the definition of singular subcontract pre-order to refine C_i in any of the i -indexed statements at the beginning of the proof. \square

In Theorem 1 we proved that all subcontract pre-order families are included in $\{\preceq_{I,O} \mid I, O \subseteq \mathcal{N}\}$; this last Theorem proves that this family of relations is also a subcontract pre-order family, thus it is the maximal one.

4.3 Subcontract relation

In the previous subsection we have introduced the input-output subcontract relation $\preceq_{I,O}$, and we have proved that it is the maximal subcontract pre-order family. Moreover, the Proposition 7 permits to abstract away from the index I of $\preceq_{I,O}$ assuming always $I = \mathcal{N}$. In this way we achieve a simpler relation \preceq_O , corresponding to $\preceq_{\mathcal{N},O}$, that we simply call *subcontract relation*, which has only one parameter indicating the set of names on which the expected contexts can perform outputs. In the definition of the subcontract relation we use $\mathcal{P}_{sys,O}$ to denote the set of processes $\mathcal{P}_{sys,\mathcal{N},O}$.

Definition 12. (Subcontract relation) *A contract C' is a subcontract of a contract C with respect to a set of output channel names $O \subseteq \mathcal{N}$, denoted $C' \preceq_O C$, if*

$$\forall P \in \mathcal{P}_{sys,O}. \quad (C \parallel P) \downarrow \Rightarrow (C' \parallel P) \downarrow$$

In order to prove that one contract C' is a subcontract of C , the Definition 12 is not exploitable due to the universal quantification on all possible parallel process P . The remainder of this subsection is devoted to the definition of an actual way for proving that two contracts are in subcontract relation. This is achieved resorting to the theory of *should-testing* [26]. The main difference of should-testing with respect to the standard must-testing [18] is that fairness is taken into account; an (unfair) infinite computation that never gives rise to success is observed in the standard must-testing scenario, while this is not the case in the should-testing scenario. The formal definition of should-testing is reported in the proof of Theorem 3.

We need a preliminary result that essentially proves that $C' \preceq_O C$ if and only if $C' \setminus \mathcal{W} - O \preceq_{\mathcal{N}} C \setminus \mathcal{W} - O$.

Lemma 1. *Let C, C' be two contracts and $O \subseteq \mathcal{N}$ be a set of output names. We have*

$$C' \preceq_O C \quad \Leftrightarrow \quad (\forall P \in \mathcal{P}_{sys}. (C \setminus \mathcal{W} - O \parallel P) \downarrow \Rightarrow (C' \setminus \mathcal{W} - O \parallel P) \downarrow)$$

Proof. Given $P \in \mathcal{P}_{sys}$, we have $(C \setminus \mathcal{W} - O \parallel P) \downarrow \Leftrightarrow (C \setminus \mathcal{W} - O \parallel P \overline{\mathcal{W} - O}) \downarrow \Leftrightarrow (C \parallel P \setminus \overline{\mathcal{W} - O}) \downarrow$. The same holds also for C' , i.e., $(C' \setminus \mathcal{W} - O \parallel P) \downarrow \Leftrightarrow (C' \setminus \mathcal{W} - O \parallel P \overline{\mathcal{W} - O}) \downarrow \Leftrightarrow (C' \parallel P \setminus \overline{\mathcal{W} - O}) \downarrow$. In the particular case of $P \in \mathcal{P}_{sys, O}$, the processes of interest in the definition of \preceq_O , we have that $P \setminus \overline{\mathcal{W} - O}$ is isomorphic to P , thus $(C \parallel P \setminus \overline{\mathcal{W} - O}) \downarrow \Leftrightarrow (C \parallel P) \downarrow$ and $(C' \parallel P \setminus \overline{\mathcal{W} - O}) \downarrow \Leftrightarrow (C' \parallel P) \downarrow$.

In the following we denote with \preceq_{test} the *should-testing* pre-order defined in [26] as follows. Intuitively, two processes P' and P are related by testing pre-order, namely $P' \preceq_{test} P$, if P' satisfies at least the same *tests* as P . A test t is any process including a special action representing success (denoted with \surd in [26], but denoted with \surd' to avoid confusion with the action \surd used in this paper to denote successful completion). Before presenting the formal definition we need to introduce the set of actions Λ as being $\Lambda = \{a, \bar{a} \mid a \in \mathcal{N}\} \cup \{\surd\}$ (i.e. we consider input and output actions and \surd : the latter is included in the set of actions of terms being tested as any other action). Similarly as in [26] we use and Λ_τ to denote $\Lambda \cup \{\tau\}$.

Formally, we write $P' \preceq_{test} P$, if $P \mathbf{shd} t$ implies $P' \mathbf{shd} t$, where $Q \mathbf{shd} t$ means that

$$\forall w \in \Lambda_\tau^*, Q'. \quad Q \parallel_\Lambda t \xrightarrow{w} Q' \quad \Rightarrow \quad \exists v \in \Lambda_\tau^*, Q'' : Q' \xrightarrow{v} Q'' \xrightarrow{\surd'}$$

where \parallel_Λ is the CSP parallel composition operator: in $R \parallel_\Lambda R'$ transitions of R and R' can be executed only if they are labeled with τ , \surd' or some $\lambda \in \Lambda$, but in this last case both R and R' must perform an action λ and their synchronization yields a transition still with label λ .

In order to resort to the setting used in [26], we define a normal form representation with terms of the process algebra considered in [26] of the finite labeled transition system (LTS) of a system P . In the following we use quadruples $(S, Lab, \longrightarrow, s_{init})$ to represent LTSes, where S is the set of states of the LTS,

Lab the set of transition labels, \longrightarrow the set of transitions with $\longrightarrow \subseteq S \times Lab \times S$ and $s_{init} \in S$ the initial state. We have that the semantics $\llbracket P \rrbracket$ of a system P is defined as being the LTS $\llbracket P \rrbracket = (S, \Lambda_\tau, \longrightarrow, P)$, where S is the set of terms P' reachable from P according to the transition relation defined by the operational rules for systems in Tables 1 and 2, i.e. such that $P \xrightarrow{w} P'$ for some (possibly empty) sequence of labels w , and \longrightarrow is the subset of such a transition relation obtained by just considering transitions between states in S .

The normal form for a system P (denoted with $\mathcal{NF}(P)$) is derived from its semantics $\llbracket P \rrbracket = (S, \Lambda_\tau, \longrightarrow, P)$ as follows, by using the operator $rec_X \theta$ (defined in [26]) that represents the value of X in the solution of the minimum fixpoint of the finite set of equations θ ,

$$\mathcal{NF}(P) = rec_{X_P} \theta \quad \text{where } \theta \text{ is the set of } S\text{-indexed equations}$$

$$X_{P'} = \sum_{(\lambda, P''): P' \xrightarrow{\lambda} P''} \lambda; X_{P''}$$

where we assume empty sums to be equal to $\mathbf{0}$, i.e. if there are no outgoing transitions from $X_{P'}$, we have $X_{P'} = \mathbf{0}$. Note that differently from our syntax, in [26] the CSP sequential composition operator $;$ is considered instead of the prefix operator we consider in this paper, thus the process $\alpha.P$ is written $\alpha; P$.

According to the definitions in [26], the semantics $\llbracket \mathcal{NF}(P) \rrbracket$ of the normal form $\mathcal{NF}(P) \equiv rec_{X_P} \theta$ is, as expected, the labeled transition system $\llbracket \mathcal{NF}(P) \rrbracket = (S', \Lambda_\tau, \longrightarrow', \mathcal{NF}(P))$, where:

- $S' = \{\mathcal{NF}(P') \equiv rec_{X_{P'}} \theta \mid P' \in S\}$
- $\longrightarrow' = \{(\mathcal{NF}(P'), \lambda, \mathcal{NF}(P'')) \mid P' \xrightarrow{\lambda} P''\}$

In the following, given a contract C , we will use $\mathcal{NF}(C)$ to stand for $\mathcal{NF}(\llbracket C \rrbracket)$. We are now in a position to define the sound characterization of the subcontract relation in terms of testing.

Theorem 3. *Let C, C' be two contracts and $O \subseteq \mathcal{N}$ be a set of output names. We have*

$$\mathcal{NF}(C' \setminus (\mathcal{N} - O)) \preceq_{test} \mathcal{NF}(C \setminus (\mathcal{N} - O)) \quad \Rightarrow \quad C' \preceq_O C$$

Proof. The proof is in Appendix C.

Note that the opposite implication

$$C' \preceq_O C \Rightarrow \mathcal{NF}(C' \setminus (\mathcal{N} - O)) \preceq_{test} \mathcal{NF}(C \setminus (\mathcal{N} - O))$$

does not hold in general. For example if we take contracts $C = a + a.c$ and $C' = b + b.c$ we have that $C' \preceq_O C$ (and $C \preceq_O C'$) for any O (there is no contract P such that $(C \parallel P) \downarrow$ or $(C' \parallel P) \downarrow$), but obviously $\mathcal{NF}(C' \setminus (\mathcal{N} - O)) \preceq_{test} \mathcal{NF}(C \setminus (\mathcal{N} - O))$ (and $\mathcal{NF}(C \setminus (\mathcal{N} - O)) \preceq_{test} \mathcal{NF}(C' \setminus (\mathcal{N} - O))$) does not hold for any O that includes $\{a, b, c\}$. As another example, consider contracts $C = \tau.\mathbf{0} + a$ and $C' = \tau.\mathbf{0} + b$. We have that $C' \preceq_O C$ (and $C \preceq_O C'$) for any O (there is

no contract P such that $(C\|P) \downarrow$ or $(C'\|P) \downarrow$, but $\mathcal{NF}(C'\|(\mathcal{N}-O)) \preceq_{test} \mathcal{NF}(C\|(\mathcal{N}-O))$ (and $\mathcal{NF}(C\|(\mathcal{N}-O)) \preceq_{test} \mathcal{NF}(C'\|(\mathcal{N}-O))$) does not hold for any O that includes $\{a, b\}$: this can be seen by considering the test $t = \surd + b; \mathbf{0}$ (and $t = \surd + a; \mathbf{0}$).

Finally, we observe that as the labeled transition system of each contract C is finite state by definition (see Definition 2), also $\mathcal{NF}(C\|(\mathcal{N}-O))$ is finite state for any O . In [26] it is proved that for finite state terms *should-testing* pre-order is decidable and an actual verification algorithm is presented. This algorithm, in the light of our Theorem 3, can be used in our setting to prove whether a contract C' is a subcontract of C (for some set of output names O). As the characterization we give in Theorem 3 is sound but not complete, it could be the case that C' is a subcontract of C even if the algorithm answers negatively thus leading to a *false negative* result. Nevertheless, all the cases of false negative we have experienced so far are related to extreme cases as those reported above, that is contracts that cannot be used in any correct system. We leave as an open problem the definition of a sound and complete characterization of the subcontract relation.

5 Conclusion

We have introduced a notion of subcontract relation useful for service oriented computing, where services are to be composed in such a way that deadlocks and livelocks are avoided. In order to be as much flexible as possible, we want to relate with our subcontract relation all those services that could safely replace their supercontracts. In the Introduction we have already discussed the practical impact of our notion of subcontract and we have compared our theory with the related literature.

Here, we add some comments about the work we have done in our previous papers, that, with respect to this paper, is also concerned with choreographic descriptions of service systems. In particular, we first perform a conceptual summary of our results (encompassing both this paper and previous papers), then we consider more detailed technical differences of this paper with respect to our previous papers.

5.1 Summary of results

Besides the problem of contract-based service retrieval that we have faced in this paper, often the design and functioning of service oriented systems, is based on high level languages called *choreography* languages in the SOC literature. Choreography languages are intended as notations for representing multi-party service compositions, that is, descriptions of the global behaviour of service-based applications in which several services reciprocally communicate in order to complete a predefined task. One of the most popular choreography languages has been developed by W3C and is called Web Services Choreography Description Language WS-CDL [27]. In WS-CDL, the basic activities in a service choreography

are interactions, that is, the atomic execution of a send and a receive operations performed by two communicating partners, called *roles*. For this reason, WS-CDL is said to follow an *interaction-oriented* approach.

When implementing an interaction-oriented choreography by assembling already available services, several mechanisms and notions need to be introduced. Often the possibility is considered of extracting, from the global specification, the behaviour of each of the involved processes in the form of a contract or of an abstract workflow. Such an extraction is often called “projection of the choreography on the roles” (see, e.g. [5]). The idea is that, based on such a projection, services are retrieved that expose a behaviour which is compatible with the extracted processes (based, e.g., on a contract refinement theory like the one presented in this paper).

We now summarize the results that we have obtained in [3, 5, 4, 6–8] about contract refinement in different scenarios.

The first mean of classification of possible scenarios is based on the *amount of knowledge* about the (initial) behavioural description of the other roles the conformance relation may depend on. We considered: knowledge about the whole choreography (full knowledge about the behaviour of other roles) or knowledge restricted to input types (receive operations) and/or output types (invoke operations) that the other roles may use. Note that in this paper we used knowledge of the latter kind. The second mean of classification of possible scenarios is based on the *kind of service compliance* assumed (i.e. of the principle assumed for assessing when multiple services work well together and form a correct system). We considered: “normal” compliance, as reported in this paper, where service interaction via invoke and receive primitives is based on synchronous handshake communication and both receive and invoke primitives may wait indefinitely (with no exception occurring) for a communication to happen (the standard CCS synchronization); “strong compliance”, where we additionally require that, whenever a service may perform an invoke on some operation, the invoked service must be already in the receive state for that operation; “queue-based compliance”, where service interaction via invoke and receive primitives is based on asynchronous communication: the receiving service puts invoke requests in an unbounded queue. Concerning service compliance we considered in all cases the fair termination property, i.e. for any finite behavioural path of the system there exists a finite path from the reached state that leads all services to successful termination. This guarantees that the system is both deadlock and, under the fairness assumption (i.e. whenever a choice is traversed infinitely often every possible outcome is chosen infinitely often), live-lock free.

Our results are summarized in the following.

- Knowledge about the whole choreography (direct conformance relation with respect to a choreography for a certain role): the maximal independent conformance relation does not exist, no matter which kind of service compliance (among those mentioned above) is considered.

- Knowledge about other initial contracts limited to input/output types they use (conformance by means of refinement of a single contract parameterized on the I/O knowledge about the others, as in this paper):
 - In the case of “normal compliance” we have that: for unconstrained contracts the maximal independent conformance relation does not exist; for contracts such that the output persistence property holds, as in the case of this paper, the maximal independent conformance relation exists and knowledge about input types is not significant; for output persistent contracts where locations expressing a unique address for every system contract are introduced and outputs are directed to a location, the maximal relation exists and knowledge about both input and output types is not significant.
 - In the case of “strong compliance” we have that: for unconstrained contracts (where outputs are directed to a location identifying a unique system contract) the maximal relation exists and knowledge about both input and output types is not significant.
 - In the case of “queue-based compliance” we have that: for unconstrained contracts (where outputs are directed to a location identifying a unique system contract) the maximal relation exists and knowledge about both input and output types is not significant.

For every maximal refinement relation above (apart from the queue-based one), we provide a sound characterization that is decidable, by resorting to an encoding into should-testing [26], a fair version of must testing. As a consequence we obtain:

- An algorithm (based on that in [26]) to check refinement.
- A classification of the maximal refinement relations with respect to existing pre-orders as, e.g., (half) bisimulation, (fair/must) testing, trace inclusion. In particular we show that the maximal refinement relations are coarser with respect to bisimulation and must testing preorders (up to some adequate encoding and treatment of fairness) in that, e.g., they allow external non-determinism on inputs to be added in refinements.

5.2 Detailed comparison with our previous work

Here, we add some comments about significant technical differences between this paper and our previous papers [5, 4, 8]. The first technical difference is that in this paper we prove the coincidence between multiple contemporaneous refinement and singular refinement, for any combination of input and output sets I , O . On the contrary, in the other papers we more simply consider the maximal sets for inputs and outputs \mathcal{N} , \mathcal{N} . This more specific result, on the one hand, required a significant technical effort (see Theorem 2 and all its preliminary results), on the other hand, allow us to achieve a more general refinement that permits also to reduce external nondeterminism. According to the theory in this paper we have, for instance, that $a + b \preceq_{\mathcal{N}-c} a + b + c$, similarly to the examples

reported at the beginning of Section 4. This kind of refinement that removes internal nondeterminism is not permitted in the other papers. Another interesting technical difference is that in this paper we use restriction in order to model the possibility to have channels that can be written by some processes and read by some other ones. In [5, 4, 8] we consider a more specific channel policy according to which channels are located, meaning that they can be read only by the processes running on the channel location. This additional constraint has an important consequence: if C' is a subcontract of C assuming a set O of possible outputs for the other contracts in the system, then C' is a subcontract of C even if we consider a larger set of outputs O' (knowledge about output types is not significant). This result does not hold in this paper as proved by the counterexample reported after Proposition 7. A final remark, is concerned with the paper [4, 8] where we present a stronger notion of compliance according to which when an output operation is ready to be executed, than the corresponding input should be already available. The interesting result is that the strong subcontract relation achieved in that paper starting from strong compliance is incomparable with the subcontract relation presented in this paper considering (standard) compliance. In fact, it is possible to show the existence of C' and C in subcontract relation according to one notion of subcontract, but not according to the other one. For instance, in this paper we have $\tau.a \preceq_{\mathcal{N}} a$, while $\tau.a$ is not strongly compliant with \bar{a} which can send an invocation on a even if the server is not yet ready to serve it. On the contrary, using the characterization of the strong subcontract relation in [4, 8], it is easy to prove that $\tau.\bar{a}.\bar{b} + \tau.\bar{b}.\bar{a}$ is a strong subcontract of $\bar{a}.\bar{b} + \bar{b}.\bar{a}$, while the former is not (standard) compliant with the contract $a.b$ which is compliant with the latter.

References

1. Allen, R., Garlan, D.: Formalizing Architectural Connection, *Proc. ICSE'94*, IEEE Computer Society Press, 1994, 71–80.
2. Autili, M., Inverardi, P., Navarra, A., Tivoli, M.: SYNTHESIS: a tool for automatically assembling correct and distributed component-based systems, *Proc. ICSE'07*, IEEE Computer Society Press, 2007, 784–787.
3. Bravetti, M., Zavattaro, G.: Contract based Multi-party Service Composition, *Proc. FSEN'07*, LNCS 4767, Springer, 2007, 207–222.
4. Bravetti, M., Zavattaro, G.: A Theory for Strong Service Compliance, *Proc. Coordination'07*, LNCS 4467, Springer, 2007, 96–112.
5. Bravetti, M., Zavattaro, G.: Towards a Unifying Theory for Choreography Conformance and Contract Compliance, *Proc. SC'07*, LNCS 4829, Springer, 2007, 34–50.
6. Bravetti, M., Zavattaro, G.: Contract Compliance and Choreography Conformance in the Presence of Message Queues, *Proc. WS-FM'08*, volume to appear of LNCS, 2008.
7. Bravetti, M., Zavattaro, G.: A Foundational Theory of Contracts for Multi-party Service Composition, *Fundamenta Informaticae* 89(4):451–478, IOS Press, 2008.
8. Bravetti, M., Zavattaro, G.: A Theory of Contracts for Strong Service Compliance, *Mathematical Structure in Computer Science*, in publication, Cambridge University Press, 2009.

9. Brogi, A., Canal, C., Pimentel, E.: Component adaptation through flexible subservicing, *Science of Computer Programming*, **63**, Elsevier, 2006, 39–56.
10. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and orchestration: A synergic approach for system design, *Proc. ICSOC'05*, LNCS 3826, Springer, 2005, 228–240.
11. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and orchestration conformance for system design, *Proc. Coordination'06*, LNCS 4038, Springer, 2006, 63–81.
12. Carbone, M., Honda, K., Yoshida, N.: Structured Communication-Centred Programming for Web Services, *Proc. ESOP'07*, LNCS 4421, Springer, 2007, 2–17.
13. Carbone, M., Honda, K., Yoshida, N., Milner, R., Brown, G., Rosst-Talbot, S.: *A Theoretical Basis of Communication-Centred Concurrent Programming*, WCD-Working Note, 2006. Available at: <http://www.dcs.qmul.ac.uk/~carbonem/cdlpaper/workingnote.pdf>
14. Carpineti, S., Castagna, G., Laneve, C., Padovani, L.: A Formal Account of Contracts for Web Services, *Proc. WS-FM'06*, LNCS 4184, Springer, 2006, 148–162.
15. Carpineti, S., Laneve, C.: A Basic Contract Language for Web Services, *Proc. ESOP'06*, LNCS 3924, Springer, 2006, 197–213.
16. Castagna, G., Gesbert, N., Padovani, L.: A Theory of Contracts for Web Services, *Proc. POPL'08*, ACM Press, 2008, 261–272.
17. De Nicola, R.: Extensional equivalences for transition systems, *Acta Informatica*, **24**(2), Springer, 1987, 211–237.
18. De Nicola, R., Hennessy, M.: Testing Equivalences for Processes, *Theoretical Computer Science*, **34**, Elsevier, 1984, 83–133.
19. Fournet, C., Hoare, C. A. R., Rajamani, S. K., Rehof, J.: Stuck-Free Conformance, *Proc. CAV'04*, LNCS 3114, Springer, 2004, 242–254.
20. Hoare, C. A. R.: *Communicating Sequential Processes*, Prentice-Hall, 1985.
21. Kobayashi, N.: Type Systems for Concurrent Processes: From Deadlock-Freedom to Livelock-Freedom, Time-Boundedness, *Proc. IFIP TCS'00*, LNCS 1872, Springer 2000, 365–389.
22. Laneve, C., Padovani, L.: The must preorder revisited - An algebraic theory for web services contracts, *Proc. Concur'07*, LNCS 4703, Springer, 2007, 212–225.
23. Mateescu, R., Poizat, P., Salaün, G.: Behavioral adaptation of component compositions based on process algebra encodings, *Proc. ASE'07*, ACM Press, 2007, 385–388.
24. Milner, R.: *Communication and Concurrency*, Prentice-Hall, 1989.
25. OASIS: *WS-BPEL: Web Services Business Process Execution Language Version 2.0*, Technical Report, OASIS, 2003.
26. Rensink, A., Vogler, W.: Fair testing, *Information and Computation*, **205**, Elsevier, 2007, 125–198.
27. W3C: *WS-CDL: Web Services Choreography Description Language*, Technical Report, W3C, 2004.

A Process algebraic representation of contracts

In this appendix we formalize the correspondence result between contracts and terms of the language \mathcal{P}_{con} by showing how to obtain from a contract $\mathcal{T} = (S, \mathcal{L}, \longrightarrow, s_h, s_0)$ a corresponding $C \in \mathcal{P}_{con}$ such that there exists a (surjective) homomorphism from the operational semantics of \mathcal{C} to \mathcal{T} itself.

Definition 13. A set of process algebraic equations is denoted by $\theta = \{X_i = C_i \mid 0 \leq i \leq n-1\}$, where n is the number of equation in the set, X_i are process variables, C_i are contract terms (possibly including free process variables). A set of process algebraic equations $\theta = \{X_i = C_i \mid 0 \leq i \leq n-1\}$ is closed if only process variables X_i , with $0 \leq i \leq n-1$, occur free in the bodies C_j , with $0 \leq j \leq n-1$, of the equations in the set.

Definition 14. Let $\mathcal{T} = (S, \mathcal{L}, \longrightarrow, s_h, s_0)$ be a contract. A contract term $C \in \mathcal{P}_{con}$ is obtained from \mathcal{T} as follows.

- Supposed $S = \{s_0, \dots, s_{n-1}\}$ (i.e. any given numbering on the states S), we first obtain from \mathcal{T} a finite closed set of equations $\theta = \{X_i = C_i \mid 0 \leq i \leq n-1\}$ as follows. Denoted by m_i the number of transitions outgoing from s_i , by α_j^i the label of the j -th transition outgoing from s_i (for any given numbering on the transitions outgoing from s_i), with $j \leq m_i$, and by $s_{succ_j^i}$ its target state, we take $C_i = \sum_{j \leq m_i} \alpha_j^i . X_{succ_j^i} + \{\mathbf{1}\}$, where $\mathbf{1}$ is present only if $s_i \xrightarrow{\lambda} s_h$ and an empty sum is assumed to yield $\mathbf{0}$.
- We then obtain, from the closed set of equations $\theta = \{X_i = C_i \mid 0 \leq i \leq n-1\}$, a closed contract term C by induction on the number of equations. The base case is $n = 1$: in this case we have that C is $recX_0.C_0$. In the inductive case we have that C is inductively defined as the term obtained from the equation set $\{X_i = C'_i \mid 0 \leq i \leq n-2\}$, where $C'_i = C_i\{recX_{n-1}.C_{n-1}/X_{n-1}\}$.

Definition 15. A homomorphism from a finite connected LTS with termination transitions $\mathcal{T} = (S, \mathcal{L}, \longrightarrow, s_h, s_0)$ to a finite connected LTS with termination transitions $\mathcal{T}' = (S', \mathcal{L}', \longrightarrow', s'_h, s'_0)$ is a function f from S to S' such that: $f(s_0) = s'_0$, $f(s_h) = s'_h$, and for all $s \in S$ we have $\{(\lambda, s') \mid f(s) \xrightarrow{\lambda} s'\} = \{(\lambda, f(s')) \mid s \xrightarrow{\lambda} s'\}$, i.e. the set of transitions performable by $f(s)$ is the same as the set of transitions performable by s when f -images of the target states are considered.

Note that, if f is a homomorphism between finite connected LTSs with finite states then f is surjective: this because all states reachable by $f(s_0)$ must be f -images of states reachable from s_0 .

Proposition 8. Let $\mathcal{T} = (S, \mathcal{L}, \longrightarrow, s_h, s_0)$ be a contract and $C \in \mathcal{P}_{con}$ be a contract term obtained from \mathcal{T} . There exists a (surjective) homomorphism from the semantics of C to \mathcal{T} itself.

Proof. Let us consider the ordering $S = \{s_0, \dots, s_{n-1}\}$ on states of S used to derive C from \mathcal{T} . We first show that every state C' in the semantics of C is such that

- 1) $C' = \mathbf{0}$ or C' is of the form $recX_i.C''$, for some $0 \leq i \leq n-1$, $C'' \in \mathcal{P}_{con}$

- 2) Every subterm of C' of the form $recX_k.C''$, for any k, C'' , is such that: $C'' = \sum_{0 \leq j \leq m} \alpha_j.C_j + \{\mathbf{1}\}$ where C_j is either of the form $recX_{succ_j}.C'_j$, for some $0 \leq succ_j \leq n-1, C'_j$, or of the form X_{succ_j} , for some $0 \leq succ_j \leq n-1$; and the following holds: $\{(\alpha, s') \mid s_k \xrightarrow{\lambda} s'\} = \{(\alpha_j, s_{succ_j}) \mid 0 \leq j \leq m\}$ and $\mathbf{1}$ is present in C'' if and only if $s_k \xrightarrow{\vee} s_h$.

Once proved this fact, the assert of the proposition is then simply derived as follows. We consider the function f from closed terms of the semantics of C to states of \mathcal{T} defined as: $f(\mathbf{0}) = s_h$ and $f(recX_i.C') = s_i$ for any i such that $0 \leq i \leq n-1$ and term C' . From property 2) above we conclude that f is an homomorphism from the semantics of C to \mathcal{T} .

The assert above on states $C' \in \mathcal{P}_{con}$ in the semantics of C is proved as follows. First we prove it to hold for C itself and we then prove that, given a contract $C_1 \in \mathcal{P}_{con}$ that satisfies it, any contract $C_2 \in \mathcal{P}_{con}$ reached by a transition from C_1 (according to the operational semantics) satisfies it.

Concerning C , we prove the assert above by showing that all equation sets θ considered when inductively obtaining C from the LTS \mathcal{T} are such that, for every term C' in the body of θ it holds:

- 1) $C' = \sum_{0 \leq j \leq m} \alpha_j.C_j + \{\mathbf{1}\}$ where C_j is either of the form $recX_{succ_j}.C'_j$, for some $0 \leq succ_j \leq n-1, C'_j$, or of the form X_{succ_j} , for some $0 \leq succ_j \leq n-1$; and the following holds: $\{(\alpha, s') \mid s_k \xrightarrow{\lambda} s'\} = \{(\alpha_j, s_{succ_j}) \mid 0 \leq j \leq m\}$ and $\mathbf{1}$ is present in C' if and only if $s_k \xrightarrow{\vee} s_h$.
- 2) Every subterm of C' of the form $recX_k.C''$, for any k, C'' , is such that: $C'' = \sum_{0 \leq j \leq m} \alpha_j.C_j + \{\mathbf{1}\}$ where C_j is either of the form $recX_{succ_j}.C'_j$, for some $0 \leq succ_j \leq n-1, C'_j$, or of the form X_{succ_j} , for some $0 \leq succ_j \leq n-1$; and the following holds: $\{(\alpha, s') \mid s_k \xrightarrow{\lambda} s'\} = \{(\alpha_j, s_{succ_j}) \mid 0 \leq j \leq m\}$ and $\mathbf{1}$ is present in C'' if and only if $s_k \xrightarrow{\vee} s_h$.

This can be easily verified by “reversed” induction on the number of equations in equation sets θ . It obviously holds for the initial equation set with n equations directly derived from \mathcal{T} : 1) directly holds by construction and 2) trivially holds because no $recX_k.C''$ subterm is present in the body of any equation. If we suppose it to hold for the equation set θ with m equations, it holds for the equation set θ' with $m-1$ equations as it can be immediately verified by considering the construction procedure of θ' from θ in the second item of Definition 14. From this we can conclude that the assert above holds for C in that C is obtained from the equation set with the single equation $X_0 = C'$ by just taking it to be $recX_0.C'$.

We finally deal with preservation of the assert above when going from a contract $C_1 \in \mathcal{P}_{con}$ to a contract $C_2 \in \mathcal{P}_{con}$. In order to prove this fact, we show, by induction on the length of the inference of transitions from C_1 to C_2 , for any $C_1, C_2 \in \mathcal{P}_{con}$, that if C_1 satisfies

- 1) C_1 is either of the form $recX_i.C''$, for some $0 \leq i \leq n-1$, $C'' \in \mathcal{P}_{con}$ or is such that: $C_1 = \sum_{0 \leq j \leq m} \alpha_j.recX_{succ_j}.C_j + \{\mathbf{1}\}$ for some $0 \leq succ_j \leq n-1$, C_j and $m \geq 0$.
- 2) Every subterm of C_1 of the form $recX_k.C''$, for any k , C'' , is such that: $C'' = \sum_{0 \leq j \leq m} \alpha_j.C_j + \{\mathbf{1}\}$ where C_j is either of the form $recX_{succ_j}.C'_j$, for some $0 \leq succ_j \leq n-1$, C'_j , or of the form X_{succ_j} , for some $0 \leq succ_j \leq n-1$; and the following holds: $\{(\alpha, s') \mid s_k \xrightarrow{\lambda} s'\} = \{(\alpha_j, s_{succ_j}) \mid 0 \leq j \leq m\}$ and $\mathbf{1}$ is present in C'' if and only if $s_k \xrightarrow{\vee} s_h$,

then C_2 satisfies the assert above. This can be easily verified by cases on the operational rule applied at the inductive step and on the operational rules with no premises, corresponding to the base case of the induction. \square

B Proof of Proposition 3

In this appendix we provide the proof of Proposition 3 that shows that a context with a single contract variable \mathcal{X} can be turned into a context of the form $[\mathcal{X}]P$. After giving some preliminary definitions we prove Lemma 2 which formalizes a result which is more general than the one stated in Proposition 3 (thus, from which Proposition 3 trivially derives).

Given a system $P \in \mathcal{P}_{sys}$ we call “reduction behaviour” of P the finite connected LTS with termination transitions $(S, \{\tau\}, \longrightarrow, s_h, P)$ defined as follows. S is the finite subset of \mathcal{P}_{sys} of systems reachable from P via (possibly empty sequences of) τ transitions, \longrightarrow are τ transitions between states of S , and s_h is the system obtained from the system P by replacing each contract with the $\mathbf{0}$ contract. We say that the reduction behaviour of a system P is homomorphic to that of a system P' if there exists an homomorphism from the reduction behaviour of P to the reduction behaviour of P' according to Definition 15.

Lemma 2. *Let \mathcal{C} be a context with a single contract variable. There exists $P \in \mathcal{P}_{sys}$ such that $I(P) = I(\mathcal{C})$, $O(P) = O(\mathcal{C})$ and, for every $C \in \mathcal{P}_{con}$, the reduction behaviour of $\mathcal{C}(C)$ is homomorphic to that of $C \parallel P$ and.*

Proof. We first observe that the reduction behaviours of two systems such that one is obtained from the other one by just exploiting commutativity and associativity of parallel composition “ \parallel ” are obviously homomorphic. Similarly the inputs and outputs offered by contexts (see Definition 8) such that one is obtained from the other one by just exploiting commutativity and associativity of parallel composition are the same. Moreover the composition of two homomorphisms yields an homomorphism.

We call \mathcal{X} the unique contract variable included in \mathcal{C} . We can assume that the topmost operator of \mathcal{C} is parallel composition. If it is a restriction we can just exploit the fact that, for every $C \in \mathcal{P}_{con}$, the reduction behaviour of $\mathcal{C}(C) = (\mathcal{C} \setminus L)(C)$ is homomorphic to that of $C'(C)$ and the inputs and outputs offered by the contexts $\mathcal{C}' \setminus L$ and \mathcal{C}' (according to Definition 8) are the same. If \mathcal{C} is directly in the form $[\mathcal{X}]$, then the assert trivially holds true for $P = [\mathbf{1}]$.

We can also assume that the contract variable \mathcal{X} occurs in \mathcal{C} in the scope of a restriction operator, otherwise the assert trivially holds by commutativity and associativity of parallel composition.

Therefore there exists a context \mathcal{C}' , a system R and a set L such that, for every $C \in \mathcal{P}_{con}$, $\mathcal{C}(C)$ can be turned into $((\mathcal{C}' \setminus L) \parallel Q)(C)$ by commutativity and associativity of parallel composition.

Moreover, we have that, for every $C \in \mathcal{P}_{con}$, the reduction behaviour of $((\mathcal{C}' \setminus L) \parallel Q)(C)$ is homomorphic to the reduction behaviour of $(\mathcal{C}' \parallel (Q \setminus \bar{L}))(C)$ and the inputs and outputs offered by the contexts $(\mathcal{C}' \setminus L) \parallel Q$ and $\mathcal{C}' \parallel (Q \setminus \bar{L})$ (according to Definition 8) are the same.

Now, if \mathcal{X} does not occur in \mathcal{C}' in the scope of a restriction operator then we are done, otherwise we consider the following function $f_{\mathcal{X}}$.

Given a context \mathcal{C}'' , such that \mathcal{X} occurs in \mathcal{C}'' in the scope of a restriction operator and a system R , $f_{\mathcal{X}}(\mathcal{C}'', R)$ is inductively defined by means of the following two cases (depending on whether \mathcal{C}'' is in the form $\mathcal{C}''' \setminus L$ or $(\mathcal{C}''' \setminus L) \parallel S$):

$$f_{\mathcal{X}}(\mathcal{C}''' \setminus L, R) = \begin{cases} \mathcal{C}''' \parallel (R \setminus \bar{L}) & \text{if } \mathcal{X} \text{ is not inside a restriction in } \mathcal{C}''' \\ f_{\mathcal{X}}(\mathcal{C}''', R \setminus \bar{L}) & \text{otherwise} \end{cases}$$

$$f_{\mathcal{X}}((\mathcal{C}''' \setminus L) \parallel S, R) = \begin{cases} \mathcal{C}''' \parallel ((S \parallel R) \setminus \bar{L}) & \text{if } \mathcal{X} \text{ is not inside a restriction in } \mathcal{C}''' \\ f_{\mathcal{X}}(\mathcal{C}''', (S \parallel R) \setminus \bar{L}) & \text{otherwise} \end{cases}$$

In the following we show that, for any context \mathcal{C}'' such that \mathcal{X} occurs in \mathcal{C}'' in the scope of a restriction operator and system R , $f_{\mathcal{X}}(\mathcal{C}'', R)$ is such that, for every $C \in \mathcal{P}_{con}$, the reduction behaviour of the system $(\mathcal{C}'' \parallel R)(C)$ is homomorphic to that of $(f_{\mathcal{X}}(\mathcal{C}'', R))(C)$; the inputs and outputs offered by the contexts $\mathcal{C}'' \parallel R$ and $f_{\mathcal{X}}(\mathcal{C}'', R)$ (according to Definition 8) are the same; and $(f_{\mathcal{X}}(\mathcal{C}'', R))(C)$ can be turned by using commutativity and associativity of parallel composition into the form $C \parallel P$, for some $P \in \mathcal{P}_{sys}$.

This result yields the thesis by taking \mathcal{C}'' to be \mathcal{C}' and R to be $Q \setminus \bar{L}$.

The result above can be shown by induction on depth of the inductive definition of function $f_{\mathcal{X}}$.

The base case is either $\mathcal{C}'' = \mathcal{C}''' \setminus L$ or $\mathcal{C}'' = (\mathcal{C}''' \setminus L) \parallel S$ with (in both cases) \mathcal{X} not occurring in \mathcal{C}''' in the scope of a restriction. In such cases the result obviously holds true because, for every $C \in \mathcal{P}_{con}$, the reduction behaviour of the system $((\mathcal{C}''' \setminus L) \parallel R)(C)$ is homomorphic to that of $(\mathcal{C}''' \parallel (R \setminus \bar{L}))(C)$ and the reduction behaviour of the system $((\mathcal{C}''' \setminus L) \parallel S \parallel R)(C)$ is homomorphic to that of $(\mathcal{C}''' \parallel ((S \parallel R) \setminus \bar{L}))(C)$. Moreover the inputs and outputs offered by the contexts $(\mathcal{C}''' \setminus L) \parallel R$ and $\mathcal{C}''' \parallel (R \setminus \bar{L})$ are the same and the inputs and outputs offered by the contexts $(\mathcal{C}''' \setminus L) \parallel S \parallel R$ and $\mathcal{C}''' \parallel ((S \parallel R) \setminus \bar{L})$ are the same.

In the inductive case the result is derived directly from the induction hypothesis and, again, from the fact that the reduction behaviour of the system $((\mathcal{C}''' \setminus L) \parallel R)(C)$ is homomorphic to that of $(\mathcal{C}''' \parallel (R \setminus \bar{L}))(C)$ and the reduction behaviour of the system $((\mathcal{C}''' \setminus L) \parallel S \parallel R)(C)$ is homomorphic to that of the system $(\mathcal{C}''' \parallel ((S \parallel R) \setminus \bar{L}))(C)$ and the inputs and outputs offered by the respective contexts are (pairwise) the same. \square

C Proof of Theorem 3

In this appendix we provide the proof of Theorem 3 stating a sound characterization of the subcontract relation in terms of the should-testing pre-order of [26]. Before reporting the proof of the theorem, we first introduce some technical machinery.

In order to build a test for the transformation $\mathcal{NF}(C \parallel (\mathcal{N} - O))$ of a contract C we have to consider a similar transformation for a system P that is executed in parallel with the contract. First of all, we consider the normal form $\mathcal{NF}(P)$ as defined above. Then, we perform the following two additional transformations that, respectively, add the \checkmark success label to the test and perform an input/output inversion so to deal with the CSP-like synchronization (where equal actions are synchronized) considered in the testing scenario of [26].

We first consider $\mathcal{NF}'(P) \equiv \mathcal{NF}(P)\{\checkmark; \checkmark'; X_{P'}/\checkmark; X_{P'} \mid P' \in \mathcal{P}_{sys}\}$, i.e. $\mathcal{NF}'(P)$ is the term $rec_{X_P}\theta'$ where θ' is obtained from θ in $\mathcal{NF}(P) \equiv rec_{X_P}\theta$ by replacing every subterm $\checkmark; X_{P'}$ occurring in θ , for any P' , with the subterm $\checkmark; \checkmark'; X_{P'}$. The LTS $\llbracket \mathcal{NF}'(P) \rrbracket = (S'', \Lambda_\tau, \longrightarrow'', \mathcal{NF}'(P))$ turns out to be, according to the definitions in [26], as follows

$$\begin{aligned} - S'' &= \{\mathcal{NF}'(P') \equiv rec_{X_{P'}}\theta' \mid P' \in S\} \cup \{\mathcal{NF}'_{\checkmark}(P'') \equiv \checkmark'.rec_{X_{P''}}\theta'' \mid \exists P' \in \\ & S: P' \xrightarrow{\checkmark} P''\} \\ - \longrightarrow'' &= \{(\mathcal{NF}'(P'), \lambda, \mathcal{NF}'(P'')) \mid P' \xrightarrow{\lambda} P'' \wedge \lambda \neq \checkmark\} \\ & \cup \{(\mathcal{NF}'(P'), \checkmark, \mathcal{NF}'_{\checkmark}(P'')) \mid P' \xrightarrow{\checkmark} P''\} \\ & \cup \{(\mathcal{NF}'_{\checkmark}(P''), \checkmark', \mathcal{NF}'(P'')) \mid \mathcal{NF}'_{\checkmark}(P'') \in S''\} \end{aligned}$$

where we assume $(S, \Lambda_\tau, \longrightarrow, P)$ to denote the LTS $\llbracket P \rrbracket$.

We then consider $\overline{\mathcal{NF}'(P)}$, i.e. the term $rec_{X_P}\theta''$ where θ'' is obtained from θ' in $\mathcal{NF}'(P) \equiv rec_{X_P}\theta'$ by turning every a occurring in θ' , for any $a \in \mathcal{N}$, into \bar{a} and every \bar{a} occurring in θ' , for any $a \in \mathcal{N}$, into a . The LTS $\llbracket \overline{\mathcal{NF}'(P)} \rrbracket = (S''', \Lambda_\tau, \longrightarrow''', \overline{\mathcal{NF}'(P)})$ turns out to be a transformation of $\llbracket \mathcal{NF}'(P) \rrbracket$ where θ'' instead of θ' is considered inside states (the state obtained in this way from a state $\mathcal{NF}'(P')$ is denoted by $\overline{\mathcal{NF}'(P')}$ and similarly a state $\mathcal{NF}'_{\checkmark}(P')$ is turned into $\overline{\mathcal{NF}'_{\checkmark}(P')}$) and whose transition labels are transformed by inverting input/output actions as described above.

We now introduce mapping of traces of $\llbracket [C] \rrbracket$ into $\llbracket \mathcal{NF}(C) \rrbracket$ and mapping of traces of $\llbracket P \rrbracket$ into $\llbracket \mathcal{NF}'(P) \rrbracket$. First of all we define a n -length trace $tr \in Tr_n^T$, with $n \geq 0$, of a LTS $\mathcal{T} = (S, Lab, \longrightarrow, s_{init})$ to be a pair (s, λ) , where s is a function from the interval of integers $[0, n]$ to states in S (we will use s_i to stand for $s(i)$) and λ is a function from the interval of integers $[1, n]$ to labels in Lab (we will use λ_i to stand for $\lambda(i)$) such that $s_{i-1} \xrightarrow{\lambda_i} s_i$, for $1 \leq i \leq n$. A n -length initial trace $tr \in ITr_n^T$ is defined in the same way with the additional constraint that $s_0 = s_{init}$. We let Tr^T to stand for $\bigcup_{n \geq 0} Tr_n^T$. In the following we will also denote a n -length trace tr simply by writing the sequence of its transitions, i.e. $tr = s_0 \xrightarrow{\lambda_1} s_1 \xrightarrow{\lambda_2} \dots \xrightarrow{\lambda_{n-1}} s_{n-1} \xrightarrow{\lambda_n} s_n$. We denote concatenation of two

traces $tr' \in Tr_n^{\mathcal{T}}$ and $tr'' \in Tr_m^{\mathcal{T}}$ such that $s'_n = s''_0$ by $tr' \widehat{\ } tr''$ defined as the trace $tr \in Tr_{n+m}^{\mathcal{T}}$ with $s_i = s'_i$ for $0 \leq i \leq n$, $\lambda_i = \lambda'_i$ for $1 \leq i \leq n$, $s_{n+i} = s''_i$ for $1 \leq i \leq m$ and $\lambda_{n+i} = \lambda''_i$ for $1 \leq i \leq m$. We also use $less_i(tr)$ to stand for the shortened trace $tr' \in Tr_{n-i}^{\mathcal{T}}$ obtained from the trace $tr \in Tr_n^{\mathcal{T}}$ by simply letting $s'_i = s_i$ for $0 \leq i \leq n-i$ and $\lambda'_i = \lambda_i$ for $1 \leq i \leq n-i$. We use $less(tr)$ to stand for $less_1(tr)$. Finally we denote with $vis(tr)$ the sequence of visible labels of the trace tr , i.e., the string $w \in (\mathcal{L} - \{\tau\})^*$ defined by induction on the length $n \geq 0$ of trace tr as follows. If $n = 0$ then $vis(tr) = \varepsilon$. If $n \geq 1$ then: $vis(tr) = vis(less(tr))$ if $\lambda_n = \tau$, $vis(tr) = vis(less(tr)) \widehat{\ } \lambda_n$ otherwise (where we use $w' \widehat{\ } w''$ to denote string concatenation).

Let us consider a transition $s \xrightarrow{\lambda} s'$ of $\llbracket C \rrbracket$. We can take $tr = s \xrightarrow{\lambda} s'$ with $tr \in Tr_1^{\llbracket C \rrbracket}$ and we define $map(tr) = \mathcal{NF}(s) \xrightarrow{\lambda} \mathcal{NF}(s')$. We then define the mapping $map(tr)$ of a whatever transition $tr = (s, \lambda) \in Tr_n^{\llbracket C \rrbracket}$ to be the transition $tr' = (s', \lambda') \in Tr_n^{\llbracket \mathcal{NF}(C) \rrbracket}$ with $s'_0 = \mathcal{NF}(s_0)$ and $s'_n = \mathcal{NF}(s_n)$ achieved by induction on $n \geq 0$ as follows. If $n = 0$ then $map(tr)$ is the trace $tr' \in Tr_0^{\llbracket \mathcal{NF}(C) \rrbracket}$ such that $s'_0 = \mathcal{NF}(s_0)$. If $n \geq 1$ then $map(tr) = map(less(tr)) \widehat{\ } map(s_{n-1} \xrightarrow{\lambda_n} s_n)$. It is immediate to verify that for any $tr \in Tr_n^{\llbracket C \rrbracket}$, $vis(map(tr)) = vis(tr)$. Moreover we have that $map : Tr_n^{\llbracket C \rrbracket} \rightarrow Tr_n^{\llbracket \mathcal{NF}(C) \rrbracket}$ is obviously injective and surjective.

Let us consider a transition $s \xrightarrow{\lambda} s'$ of $\llbracket P \rrbracket$. We can take $tr = s \xrightarrow{\lambda} s'$ with $tr \in Tr_1^{\llbracket P \rrbracket}$. We define $map_{\checkmark}(tr)$ as follows. If $\lambda \neq \checkmark$ then $map_{\checkmark}(tr) = \mathcal{NF}'(s) \xrightarrow{\lambda} \mathcal{NF}'(s')$. Otherwise, if $\lambda = \checkmark$ then $map_{\checkmark}(tr) = \mathcal{NF}'(s) \xrightarrow{\checkmark} \mathcal{NF}'_{\checkmark}(s')$. We then define the mapping $map_{\checkmark}(tr)$ of a whatever transition $tr = (s, \lambda) \in Tr_n^{\llbracket P \rrbracket}$ (tr may include \checkmark only as the final transition because target states of \checkmark transitions have no outgoing transitions in the semantics of systems) to be the transition $tr' = (s', \lambda') \in Tr_n^{\llbracket \mathcal{NF}'(P) \rrbracket}$ with $s'_0 = \mathcal{NF}'(s_0)$ and, in the case $n \geq 1$, $s'_n = \mathcal{NF}'(s_n)$ if $\lambda_n \neq \checkmark$, $s'_n = \mathcal{NF}'_{\checkmark}(s_n)$ otherwise, achieved by induction on $n \geq 0$ as follows. If $n = 0$ then $map_{\checkmark}(tr)$ is the trace $tr' \in Tr_0^{\llbracket \mathcal{NF}'(P) \rrbracket}$ such that $s'_0 = \mathcal{NF}'(s_0)$. If $n \geq 1$ then $map_{\checkmark}(tr) = map_{\checkmark}(less(tr)) \widehat{\ } map_{\checkmark}(s_{n-1} \xrightarrow{\lambda_n} s_n)$. It is immediate to verify that for any $tr \in Tr_n^{\llbracket P \rrbracket}$, $vis(map_{\checkmark}(tr)) = vis(tr)$. Moreover we have that $map_{\checkmark} : Tr_n^{\llbracket P \rrbracket} \rightarrow Tr_n^{\llbracket \mathcal{NF}'(P) \rrbracket}$ is injective (because the last transition of the trace singles out at each inductive step a unique mapping that can produce it) and is surjective over the codomain of traces that begin with a state of the form $\mathcal{NF}'(s)$ with $s \in \llbracket P \rrbracket$ and do not include \checkmark transitions.

We finally define some other auxiliary functions that will be used in the proof. Given a trace $tr \in Tr_n^{\llbracket \mathcal{NF}'(P) \rrbracket}$ or $tr \in Tr_n^{\llbracket \mathcal{NF}'(P) \rrbracket}$, we define \overline{tr} to be $tr' = (s', \lambda')$ defined as: $s'_i = \overline{s}_i$ and $\lambda'_i = \overline{\lambda}_i$ for any i (where the application of the overbar to states or labels that have it already causes its removal and it has no effect when applied to τ and \checkmark labels). Notice that $vis(\overline{tr}) = vis(tr)$ denoting the application of the overbar to any label occurring in the sequence of visible labels.

Proof of Theorem 3:

According to the definition of should-testing of [26], since

$$\mathcal{NF}(C' \parallel (\mathcal{N}-O)) \preceq_{test} \mathcal{NF}(C \parallel (\mathcal{N}-O))$$

we have that, for every test t , if $\mathcal{NF}(C \parallel (\mathcal{N}-O)) \mathbf{shd} t$, then also $\mathcal{NF}(C' \parallel (\mathcal{N}-O)) \mathbf{shd} t$, where $Q \mathbf{shd} t$ iff

$$\forall w \in \Lambda_r^*, Q'. \quad Q \parallel_A t \xrightarrow{w} Q' \quad \Rightarrow \quad \exists v \in \Lambda_r^*, Q'' : Q' \xrightarrow{v} Q'' \xrightarrow{v'}$$

where \parallel_A is the CSP parallel operator.

Let us now consider $P \in P_{sys}$ with $(C \parallel \mathcal{N} - O \parallel P) \downarrow$. In the following we will provide a first subproof that this implies $\mathcal{NF}(C \parallel (\mathcal{N}-O)) \mathbf{shd} \overline{\mathcal{NF}'(P)}$. Since $\mathcal{NF}(C' \parallel (\mathcal{N}-O)) \preceq_{test} \mathcal{NF}(C \parallel (\mathcal{N}-O))$, from this we can derive $\mathcal{NF}(C' \parallel (\mathcal{N}-O)) \mathbf{shd} \overline{\mathcal{NF}'(P)}$. In the following we will provide a second subproof that this implies $(C' \parallel \mathcal{N} - O \parallel P) \downarrow$. The thesis, then, directly follows from Lemma 1.

First subproof: $(\tilde{C} \parallel P) \downarrow \Rightarrow \mathcal{NF}(\tilde{C}) \mathbf{shd} \overline{\mathcal{NF}'(P)}$, with $\tilde{C} = C \parallel (\mathcal{N}-O)$.

We consider the trace $tr = (s, \lambda) \in ITr_r^{\llbracket \mathcal{NF}(\tilde{C}) \parallel_A \overline{\mathcal{NF}'(P)} \rrbracket}$ such that $\lambda_i \neq \checkmark$ for any i . There exist $\dot{tr} \in ITr_n^{\llbracket \mathcal{NF}(\tilde{C}) \rrbracket}$ and $\ddot{tr} \in ITr_m^{\llbracket \overline{\mathcal{NF}'(P)} \rrbracket}$, corresponding to the local moves performed by the two parallel processes when doing trace tr , such that $vis(\dot{tr}) = vis(\ddot{tr})$.

We have two cases for the structure of the last state of trace \ddot{tr} :

1. $\ddot{s}_m \equiv \overline{\mathcal{NF}'(s)}$ for some $s \in [P]$
2. $\ddot{s}_m \equiv \mathcal{NF}'_{\checkmark}(s)$ for some $s \in [P]$

Let us start by taking \ddot{tr} to be as in the simpler case (2). Since, by def. of $\mathcal{NF}'(P)$, $\ddot{s}_m = \mathcal{NF}'_{\checkmark}(s) \Rightarrow \ddot{s}_m \xrightarrow{\checkmark}$, we have $s_r = \dot{s}_n \parallel_A \ddot{s}_m \xrightarrow{\checkmark}$ and we are done.

We now move to the non-trivial case (1) for \ddot{tr} . Consider $\dot{tr}' = map_{\checkmark}^{-1}(\ddot{tr}) \in ITr_m^{\llbracket P \rrbracket}$. We have $\ddot{s}_m = \overline{\mathcal{NF}'(\dot{s}'_m)}$. Let us also consider $tr' = map^{-1}(\dot{tr}) \in ITr_n^{\llbracket [\tilde{C}] \rrbracket}$. We have $\dot{s}_n = \mathcal{NF}(\dot{s}'_n)$. Moreover, $vis(\dot{tr}') = vis(\ddot{tr}) = vis(\dot{tr}) = vis(\dot{tr}')$. Therefore, there exists $tr'' \in ITr_r^{\llbracket [\tilde{C}] \parallel P \rrbracket}$, with $\lambda'_i = \tau$ for any i , such that $s'_r = \dot{s}'_n \parallel \ddot{s}'_m$.

Since $(\tilde{C} \parallel P) \downarrow$ we have that there exists $tr'' \in Tr_{r''}^{\llbracket [\tilde{C}] \parallel P \rrbracket}$ with $s''_0 = s'_r$, $\lambda''_i = \tau$ for $1 \leq i \leq r'' - 1$ and $\lambda''_{r''} = \checkmark$. Therefore, there exist $\dot{tr}'' \in Tr_{n''}^{\llbracket [\tilde{C}] \rrbracket}$ with $\dot{s}''_0 = \dot{s}'_n$ and $\ddot{tr}'' \in Tr_{m''}^{\llbracket P \rrbracket}$, with $\ddot{s}''_0 = \ddot{s}'_m$ corresponding to the local moves performed by the two parallel processes when doing trace tr'' , such that $vis(\dot{tr}'') = vis(\ddot{tr}'')$.

Let us now consider $\dot{tr}''' = map(\dot{tr}'') \in Tr_{n''}^{\llbracket \mathcal{NF}(\tilde{C}) \rrbracket}$. We have $\dot{s}'''_0 = \mathcal{NF}(\dot{s}''_0) = \mathcal{NF}(\dot{s}'_n)$ and $\dot{\lambda}'''_{n''} = \checkmark$. Let us also consider $\ddot{tr}''' = map_{\checkmark}(\ddot{tr}'') \in Tr_{m''}^{\llbracket \overline{\mathcal{NF}'(P)} \rrbracket}$. We have $\ddot{s}'''_0 = \overline{\mathcal{NF}'(\dot{s}''_0)} = \overline{\mathcal{NF}'(\dot{s}'_m)}$ and $\ddot{\lambda}'''_{m''} = \checkmark$. Moreover, $vis(\dot{tr}''') = vis(\dot{tr}'') = vis(\dot{tr}''')$.

Therefore, there exists $tr''' \in Tr_{r''}^{[\mathcal{NF}(\tilde{C}) \parallel_A \overline{\mathcal{NF}'(P)}]}$. Such trace has initial state $s_0''' = \mathcal{NF}(\dot{s}'_n) \parallel_A \overline{\mathcal{NF}'(\dot{s}'_m)} = \dot{s}_n \parallel_A \ddot{s}_m = s_r$, and $s_{r''}''' = \dot{s}'_{n''} \parallel_A \ddot{s}'_{m''}$. Moreover, since, by def. of $\mathcal{NF}'(P)$, $\ddot{\lambda}'_{m''} = \sqrt{\Rightarrow} \ddot{s}'_{m''} \xrightarrow{\sqrt{\cdot}}$, we have $s_{r''}''' \xrightarrow{\sqrt{\cdot}}$.

Second subproof: $\mathcal{NF}(\tilde{C}') \text{ shd } \mathcal{NF}'(P) \Rightarrow (\tilde{C}' \parallel P) \downarrow$, with $\tilde{C}' = C' \setminus (\mathcal{N} - O)$.

We consider the trace $tr = (s, \lambda) \in ITr_r^{[\tilde{C}' \parallel P]}$ such that $\lambda_i = \tau$ for any i . There exist $\dot{tr} \in ITr_n^{[[\tilde{C}']]}$ and $\ddot{tr} \in ITr_m^{[P]}$ corresponding to the local moves performed by the two parallel processes when doing trace tr , such that $\overline{vis(\dot{tr})} = \overline{vis(\ddot{tr})}$.

Let us now consider $\dot{tr}' = \overline{map(\dot{tr})} \in ITr_n^{[\mathcal{NF}(\tilde{C}')]}$. We have $\dot{s}'_n = \mathcal{NF}(\dot{s}_n)$. Let us also consider $\ddot{tr}' = \overline{map_{\sqrt{\cdot}}(\ddot{tr})} \in ITr_m^{[\mathcal{NF}'(P)]}$. We have $\ddot{s}'_m = \overline{\mathcal{NF}'(\ddot{s}_m)}$ because \ddot{tr} does not include a $\sqrt{\cdot}$ transition. Moreover, $\overline{vis(\dot{tr}')} = \overline{vis(\dot{tr})} = \overline{vis(\ddot{tr})} = \overline{vis(\ddot{tr}')}$. Therefore, there exists $tr' \in ITr_r^{[\mathcal{NF}(\tilde{C}') \parallel_A \overline{\mathcal{NF}'(P)}]}$ with $s'_r = \dot{s}'_n \parallel_A \ddot{s}'_m = \mathcal{NF}(\dot{s}_n) \parallel_A \overline{\mathcal{NF}'(\ddot{s}_m)}$.

Since $\mathcal{NF}(\tilde{C}') \text{ shd } \mathcal{NF}'(P)$ we have that there exists $tr'' \in Tr_{r''}^{[\mathcal{NF}(\tilde{C}') \parallel_A \overline{\mathcal{NF}'(P)}]}$ with $s''_0 = s'_r$, such that $\lambda''_i \neq \sqrt{\cdot}$ for $1 \leq i \leq r'' - 1$ and $\lambda''_{r''} = \sqrt{\cdot}$.

There exist $\dot{tr}'' \in Tr_{n''}^{[\mathcal{NF}(\tilde{C}')]}$ with $\dot{s}''_0 = \dot{s}'_n$ and $\ddot{tr}'' \in Tr_{m''}^{[\mathcal{NF}'(P)]}$, with $\ddot{s}''_0 = \ddot{s}'_m$ corresponding to the local moves performed by the two parallel processes when doing trace tr'' . We must have $\ddot{\lambda}''_{m''} = \sqrt{\cdot}$ and, since $\ddot{s}''_0 = \ddot{s}'_m$ is in the form $\overline{\mathcal{NF}'(\ddot{s}_m)}$, by def. of $\mathcal{NF}'(P)$ we have ($r'' \geq 2$ and) $\ddot{\lambda}''_{m''-1} = \sqrt{\cdot}$. Moreover we must have $\overline{vis(\dot{tr}'')} = \overline{vis(\text{less}(\ddot{tr}''))}$, hence in particular $\dot{\lambda}''_{n''} = \sqrt{\cdot}$: the $\sqrt{\cdot}$ transition must be the last one in such a trace (i.e. there are no τ transitions afterward) because target states of $\sqrt{\cdot}$ transition have no outgoing transitions in the semantics of contracts and the transformation $\mathcal{NF}(\tilde{C}')$ cannot add outgoing τ transitions to such states.

Let us now consider $\dot{tr}''' = \overline{map^{-1}(\dot{tr}'')} \in Tr_{n''}^{[[\tilde{C}']]}$. We have $\dot{s}''_0 = \dot{s}'_n = \mathcal{NF}(\dot{s}'''_0)$ and $\dot{\lambda}''_{n''} = \sqrt{\cdot}$. Let us also consider $\ddot{tr}''' = \overline{map_{\sqrt{\cdot}}^{-1}(\ddot{tr}'')} \in Tr_{m''-1}^{[P]}$. We have $\ddot{s}''_0 = \ddot{s}'_m = \overline{\mathcal{NF}'(\ddot{s}'''_0)}$ and $\ddot{\lambda}''_{m''-1} = \sqrt{\cdot}$. Moreover, $\overline{vis(\ddot{tr}''')} = \overline{vis(\text{less}(\dot{tr}''))} = \overline{vis(\dot{tr}'')} = \overline{vis(\dot{tr}''')}$.

Therefore, there exists $tr''' \in Tr_{r''}^{[\tilde{C}' \parallel P]}$ such that $\lambda'''_i = \tau$ for $1 \leq i \leq r'' - 1$, with $s_0''' = \dot{s}'''_0 \parallel_A \ddot{s}'''_0 = \dot{s}_n \parallel_A \ddot{s}_m = s_r$ and $\lambda'''_{r''} = \sqrt{\cdot}$. \square