Fundamenta Informaticae XX (2008) 1–28
IOS Press

A Foundational Theory of Contracts for Multi-party Service Composition*

Mario Bravetti

Gianluigi Zavattaro

Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura Anteo Zamboni 7, I-40127 Bologna, Italy.
e-mail:{bravetti, zavattar}@cs.unibo.it

Abstract. In the context of Service Oriented Computing, contracts are descriptions of the observable message-passing behavior of services. Contracts have been already successfully exploited to solve the problem of client/service composition. Inspired by current orchestration languages, we consider services where the choice to perform an output may not depend on the environment. Under this assumption, we propose a new theory of contracts which also addresses the problem of composition of multiple services (not only one client with one service). Moreover, we relate our theory of contracts with the theory of must testing pre-order (interpreted as a subcontract relation) and we show that a compliant group of contracts is still compliant if every contract is replaced by one of its subcontracts.

1. Introduction

Service Oriented Computing (SOC) is a novel 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. One of the peculiarities of Service Oriented Computing, distinguishing it from other distributed computing paradigms (such as component based software engineering), is that it is centered around the so-called *message oriented architectures*. This means that, given a set of collaborating services, the current state of their interaction is stored inside the exchanged messages and not only within the services. From a practical viewpoint, this means that it is necessary

*Revised and extended version of [3]. Research partially funded by EU Integrated Project Sensoria, contract n. 016004.

to include, in the exchanged messages, the so-called correlation information that permits a service to associate a received message to the correct session of interaction (in fact, the same service could be contemporaneously involved in different sessions at the same time).

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 behaviors are complementary. In particular, it is important to check whether in a set of services, combined in order to collaborate, no service waits indefinitely for a never incoming message.

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 behavior. In the service oriented computing literature, this kind of information is referred to as the *service contract* [12]. 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.

Compliance checking based on the behavioral descriptions of the composed entities has been already considered, for instance, in the context of component-based systems (see e.g. [6, 2, 20]) or for client-service interaction [11]. 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 behavior of one component or deal with possible mismatchings 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* [24], which contrasts with the *orchestrated* approach [22] 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 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 *subcontract*.

Hence, the focus is not in the introduction of a new calculus for SOC, but on studying the influence of the very basic properties (external choice between outputs, mixed choice etc...) of a calculus for contracts (which is as simple as possible but does not make decidability trivial, i.e. can express recursive behaviours) on the notion of subcontract. The idea is that this study is needed and forms a theoretical starting point for richer calculi for SOC.

We foresee at least two main applications for our theory of contracts. On the one hand, it can be exploited in the *service discovery* phase. Consider, for instance, a service system defined in terms of the contracts that should be exposed 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 subcontract. On the other hand, the notion of subcontract could be useful in *service updates* in order 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 the paper is the definition of a theory of contracts achieved following a general, novel approach. This approach is based on the following simple intuition: as a contract is an abstraction of a service behavior, we can also consider the relationship from the opposite point of view, that is, considering a service behavior as a concretion/refinement of a contract. If we are able to capture the correct notion of refinement (assuming it is a pre-order), we can associate to a service its contract simply by taking a canonical representative of the corresponding equivalence class induced by the refinement. In this way, we reduce the problem of finding the right information to be exposed in a contract to the definition of a suitable refinement. The approach we take to define our refinement is to first consider the property that we want a "good" refinement should preserve, and then consider the maximum of such refinements. This is similar to the co-inductive approach considered, e.g., in the definition of the bisimulation relation for CCS [21].

More formally, we present two languages: the former for the specification of both services and contracts, the latter used to specify their composition. We use the second calculus 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 refinement, 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 calculus that we propose, the subcontract relation achieved according to this approach is actually the largest subcontract pre-order. In fact, in other theories of contracts recently proposed in the literature (details are reported in the next subsection), this property does not hold. This property is important because, as already discussed above, we foreseen the exploitation of our theory of contracts for the independent retrieval of services in multi-party compositions.

The calculus for contracts that we propose is similar to traditional process calculi distinguishing between deadlock and successful termination. This distinction is necessary in order to model the fact that a service could internally deadlock due to its internal parallelism. Another peculiarity of the calculus is

that the decision to execute output actions may not depend on the other services in the system. This reflect the fact that in asynchronous distributed systems, such as the Internet currently used as transport layer for Web Services, there is an implicit asymmetry between input and output actions. This asymmetry derives from the fact that an output operation consists of the emission of a message, while an input operation coincides with the reception of a message. The decision to send a message in an asynchronous system is taken locally, and cannot be forbidden by the remote expected receiver. Moreover, as we will see, the existence of maximal independent refinement, that allows us to refine contracts independently, is a consequence of such an asymmetry between inputs and outputs.

In more technical terms, we consider a semantics for output actions (denoted with an overlined action name such as $\overline{b}$) which is different from that of input actions (denoted simply with an action names such as a). The semantics of input is, as usual, represented by a transition labeled with the corresponding action name. In the semantics of output, on the contrary, we consider two subsequent transitions, one labeled with τ (that denotes an internal actions representing the decision to execute the output actions) and one labeled with the overlined corresponding action name (that represents the actual execution of the output action). According to this semantics, the mixed choice $a+\overline{b}$ between an input and an output action is actually a choice between the input action a and the internal τ transition representing the decision to execute the output $\overline{b}$.

Another important technical achievement of the paper is a characterization of the subcontract relation in a testing-like scenario [15]: 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* [23] 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 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 relevant differences between our form of testing and the traditional one proposed by De Nicola-Hennessy [15]. The most relevant 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 relevant 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 [23]. 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 [17] coincide (at least for finitely branching processes without divergences) [14]. 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

[1] We use $\mathbf{0}$ to denote unsuccessful termination, $\mathbf{1}$ for successful completion and $\_;\_$ for sequential composition.

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 behaviors of refined services. Nevertheless, the asymmetry that we have between input and output actions allows us to replace a service with another one having additional behavior.

Contracts have been initially introduced in the context of process calculi by Fournet et al. [16]. As far as service oriented computing is concerned, an initial theory of contracts for client-service interaction has been proposed by Carpineti et al. [11] and then independently extended along different directions by Bravetti and Zavattaro (in a preliminary version of this paper [3] as well as in two other papers [4, 5]), by Laneve and Padovani [19], and by Castagna et al. [13]

In [16] 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 [16] 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 [16]. Another difference follows from the exploitation of the restriction $(\nu\tilde{a})$; this is used in [16] 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 behavior of a service, but the flow of execution of its operations inside one session of communication.

The closed-world assumption is considered also in [11] where, as in our case, a service oriented scenario is considered. In particular, in [11] 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 [11] where it is not in general possible to substitute a service exposing one contract with another one exposing a subcontract. Another relevant difference is that the contracts in [11] comprises also mixed choices.

The preliminary version of this paper [3] introduces several interesting new aspects not considered in the initial approach of Carpineti et al. For instance, we consider also contracts with an infinite behavior admitting the repetition operator, 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 behaviors are permitted. For instance, the three contracts

$$\overline{a};\overline{b} \qquad a;(\overline{c};d)^* \qquad (c;\overline{d})^*;b$$

($\_^*$ denotes repetition) 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. [7, 8] and by Carbone et al. [9]. 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. [11] discussed above has been extended by (some of) the original authors in two ways, in [19] by explicitly associating to a contract the considered input/output alphabet, in [13] by associating to services a dynamic filter which eliminates from the service behavior those interactions that are not admitted by the considered contract.

The explicit information about the input/output alphabet used in [19] 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 [19], does not perm it to achieve one of the most interesting property we prove in this paper, that is, independent contract refinement. In fact, according to [19] 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 [19] does not admit parallel discovery of services in multi-party service systems.

The dynamic filters of [13], on the contrary, allow for independent refinement, at the price of synthesizing a specific filter used to eliminate the additional behaviors 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 on session types (e.g. that in [18] and [10]) gives rise to notions of refinement in, somehow restricted, scenarios that can be syntactically characterized. For instance, the work in [18] 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 [10] 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 one is incomparable because it deals with very special cases and in our approach, e.g., $a|b$ does not refine $a;b$. More precisely we just consider knowledge about types of inputs and outputs of other contracts and not about their syntactical structure.

1.3. Structure of the paper.

Section 2 reports syntax and semantics of the process calculi. In Section 3 we describe our theory of contracts and we prove our main results. Finally, Section 4 reports some conclusive remarks.

This paper is an extended version of [3] that additionally includes the full proofs for all theorems,

some explanatory examples and a simple case study, and a more detailed comparison with the related literature.

2. Syntax and Semantics of the Process Calculi

In this Section we introduce incrementally the two calculi. The first one is the calculus for contracts; it is a typical calculus comprising two possible final states (failure or success), input and output prefixes, sequencing, choice, parallel composition, restriction and repetition. Differently from standard process calculi we distinguish between internal and external actions. This distinction is a consequence of the separation between the calculus for contracts and that for contract compositions: local actions are those that can be executed locally within the same contract, while external actions are those that can be executed between two distinct composed contracts. This corresponds to the distinction between local synchronization, by means of internal *links*, and remote interaction, by means of *operation invocations*, that we have in languages for Web Services compositions such as WS-BPEL.

2.1. Definition of Contracts

We assume a denumerable set of names $\mathcal{N} = \{a, b, c, ...\}$. The set $\mathcal{N}_{con} = \{a_* \mid a \in \mathcal{N}\}$ is the set of local names. We define $\mathcal{N}_{all} = \mathcal{N} \cup \mathcal{N}_{con}$. The set $\mathcal{A} = \mathcal{N} \cup \{\overline{a} \mid a \in \mathcal{N}\}$ is the set of input and output actions. The set $\mathcal{A}_{con} = \mathcal{N}_{con} \cup \{\overline{a}_* \mid a_* \in \mathcal{N}_{con}\}$ is the set of input and output local actions. We take β to range over the set of all actions $Act = \mathcal{A}_{con} \cup \mathcal{A} \cup \{\tau\}$, where τ denotes an internal computation.

Definition 2.1. (Contracts) The syntax of contracts is defined by the following grammar

$$\begin{array}{rcl} C & ::= & \mathbf{0} \mid \mathbf{1} \mid \tau \mid a_* \mid \overline{a}_* \mid a \mid \overline{a} \mid <\overline{a}> \mid \\ & & C;C \mid C+C \mid C|C \mid C\backslash M \mid C^* \end{array}$$

where $M \subseteq \mathcal{N}_{con}$.

We consider eight possible atoms: unsuccessful termination $\mathbf{0}$, successful termination $\mathbf{1}$, silent move τ, internal input action a_*, internal output action $\overline{a}_*$ and external input action a, external output action $\overline{a}$ and external output action $<\overline{a}>$ that represents an output "$\overline{a}$" which has been decided to be performed but has not synchronized yet with a receiving "a".

The operators are: sequencing $\_;\_$, choice $\_+\_$, parallel $\_|\_$, restriction $\_ \setminus M$, and repetition $\_^*$. We use C, C', $\cdots$, D, D', $\cdots$ and E, E', $\cdots$ to range over contract terms. The set of "initial" contracts C, i.e. contracts in which outputs $<\overline{a}>$ do not occurr, is denoted by $\mathcal{P}_{con}$.

In the following we will omit trailing "$\mathbf{1}$" when writing contracts and, given a set of names M, with $\overline{M} = \{\overline{a} \mid a \in M\}$ we denote the set of output actions performable on those names. Moreover, when a set $\{a\}$ contains only one element, namely a, we denote it simply with a.

The operational semantics of contracts is defined by the rules in Table 2.1 (plus the omitted symmetric rules). We take λ to range over the set of labels $\mathcal{L} = Act \cup \{\surd\}$, where $\surd$ denotes successful termination. In the following we will use $\mathcal{P}_{conSt}$ to denote the set of contract states, defined as the terms that are reachable by a (possibly empty) sequence of transitions from initial contracts in $\mathcal{P}_{con}$.

$$\mathbf{1} \xrightarrow{\surd} \mathbf{0} \qquad \beta \xrightarrow{\beta} \mathbf{1} \quad \beta \notin \{\overline{a} \,|\, a \in \mathcal{N}\} \qquad \overline{a} \xrightarrow{\tau} <\overline{a}> \qquad <\overline{a}> \xrightarrow{\overline{a}} \mathbf{1}$$

$$\frac{C \xrightarrow{\lambda} C'}{C+D \xrightarrow{\lambda} C'} \qquad \frac{C \xrightarrow{\lambda} C' \quad \lambda \neq \surd}{C;D \xrightarrow{\lambda} C';D} \qquad \frac{C \xrightarrow{\surd} C' \quad D \xrightarrow{\lambda} D'}{C;D \xrightarrow{\lambda} D'}$$

$$\frac{C \xrightarrow{a_*} C' \quad D \xrightarrow{\overline{a}_*} D'}{C|D \xrightarrow{\tau} C'|D'} \qquad \frac{C \xrightarrow{\surd} C' \quad D \xrightarrow{\surd} D'}{C|D \xrightarrow{\surd} C'|D'} \qquad \frac{C \xrightarrow{\lambda} C' \quad \lambda \neq \surd}{C|D \xrightarrow{\lambda} C'|D}$$

$$\frac{C \xrightarrow{\lambda} C' \quad \lambda \notin M \cup \overline{M}}{C\backslash M \xrightarrow{\lambda} C'\backslash M} \qquad C^* \xrightarrow{\surd} \mathbf{0} \qquad \frac{C \xrightarrow{\lambda} C'}{C^* \xrightarrow{\lambda} C';C^*}$$

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

The operational semantics is rather standard for process calculi with sequential composition, where the $\surd$ label is used to explicitly denote completion. The unique relevant remark is that synchronization within a contract is permitted only on local names a_*; the synchronization on global names between different contracts will be considered in the next calculus used to model the composition of 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 $w = \lambda_1\lambda_2 \cdots \lambda_{n-1}\lambda_n$ (possibly empty, i.e., $w = \varepsilon$), we use $C \xrightarrow{w} C'$ to denote the sequence of transitions $C \xrightarrow{\lambda_1} C_1 \xrightarrow{\lambda_2} \cdots \xrightarrow{\lambda_{n-1}} C_{n-1} \xrightarrow{\lambda_n} C'$ (in case of $w = \varepsilon$ we have $C' = C$, i.e., $C \xrightarrow{\varepsilon} C$).

We now formalize an important property of our calculus that we call *output persistence*. This 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. 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 3.2 –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 **??**.

In order to formally prove the output persistence property we need to prove two preliminary lemmata.

Lemma 2.1. Let $C \in \mathcal{P}_{conSt}$ be a contract state. Then, if $C \xrightarrow{\overline{a}}$ then $C \not\xrightarrow{\surd}$.

Proof:
By induction on the length of the proof of $C \xrightarrow{\overline{a}} C'$.

The base case is trivial as $C =< \overline{a} >$, thus also $C \not\xrightarrow{\surd}$. In the inductive case we proceed by case analysis on the last used rule. We first observe that, in general, if $E \xrightarrow{\overline{a}}$ then E cannot be an initial contract belonging to $\mathcal{P}_{con}$. This because in terms of $\mathcal{P}_{con}$ $\overline{a}$ transitions are always preceded by an

outgoing τ transition. Therefore, the transition $C \xrightarrow{\overline{a}}$ derives in one step from a transition $\overline{a}$ of a non-initial E (which is also a subterm of C). As E is non-initial and it must be a derivative of some initial contract, i.e. $E \in \mathcal{P}_{conSt}$, there are only three possible cases: (i) the first rule for sequential composition, (ii) the last rule for parallel composition, and (iii) the rule for restriction.

In case (i) we have that $C = E; E'$ with $E \xrightarrow{\surd}\!\!\!\!\!/\;\;$ by inductive hypothesis. Therefore, the actions of E' cannot be executed by $E; E'$, thus also $E; E' \xrightarrow{\surd}\!\!\!\!\!/\;\;$. In case (ii) we have that $C = E|E'$ with $E \xrightarrow{\surd}\!\!\!\!\!/\;\;$ by inductive hypothesis. Therefore, also $E|E' \xrightarrow{\surd}\!\!\!\!\!/\;\;$ as transitions labeled with $\surd$ require synchronization. In case (iii), the lemma follows directly form the inductive hypothesis. □

Lemma 2.2. Let $C \in \mathcal{P}_{conSt}$ be a contract state. such that $C \xrightarrow{\beta} C'$ with $\beta \neq \overline{a}$. If $C \xrightarrow{\overline{a}}$ then $C' \xrightarrow{\overline{a}}$.

Proof:
By induction on the length of the proof of $C \xrightarrow{\beta} C'$.

In the base case we have that $C = \beta$. As $\beta \neq \overline{a}$, then $C \xrightarrow{\overline{a}}\!\!\!\!\!/\;\;$. In the inductive case we proceed by case analysis on the last used rule. We report the analysis only of some significant cases.

If the last used rule is the one for choice, we have that C is an initial contract of $\mathcal{P}_{con}$ thus (as discussed in the proof of the previous lemma) we have that $C \xrightarrow{\overline{a}}\!\!\!\!\!/\;\;$.

If the last used rule is the first one for sequential composition we have $C = E; E'$ and the transition is inferred by E. If $E; E' \xrightarrow{\overline{a}}$ then we also have $E \xrightarrow{\overline{a}}$ because E' is an initial contract (thus $E' \xrightarrow{\overline{a}}\!\!\!\!\!/\;\;$). The thesis follows directly from the inductive hypothesis.

If the last used rule is the second one for sequential composition we have $C = E; E'$ and $E \xrightarrow{\surd}$. By previous lemma we have that $E \xrightarrow{\overline{a}}\!\!\!\!\!/\;\;$ and moreover E' is an initial contract (thus $E' \xrightarrow{\overline{a}}\!\!\!\!\!/\;\;$). Henceforth, $E; E' \xrightarrow{\overline{a}}\!\!\!\!\!/\;\;$.

If the last used rule is the last one for parallel composition, we have $C = E|E'$ and the transition is inferred by E. If $C \xrightarrow{\overline{a}}$, then either $E \xrightarrow{\overline{a}}$ or $E' \xrightarrow{\overline{a}}$. In the first case the thesis follows from the inductive hypothesis. In the second case the thesis follows from the fact that E' is kept as parallel subterm also in the target of the transition. □

Proposition 2.1. (Output persistence) Let $C \in \mathcal{P}_{con}$ be a contract such that $C \xrightarrow{w} C' \xrightarrow{\overline{a}}$. We have that, for every C'' such that $C' \xrightarrow{w'} C''$ and $C'' \xrightarrow{\surd}$, the string w' must include $\overline{a}$.

Proof:
Direct consequence of Lemma 2.1 and Lemma 2.2. □

2.2. Composing Contracts

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. In our specific case the $[C]$ operator is required also for technical reasons to define the scope of the local actions of C (it corresponds to an implicit restriction on all local names).

Besides the parallel composition operator $\|$, we consider also restriction $\backslash\!\backslash$ 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 $[C]\backslash\!\backslash\{a, \overline{b}\}$ 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 $\overline{a}$):

$$[C_1]\backslash\!\backslash\overline{a} \;\|\; [C_2]\backslash\!\backslash a \;\|\; [C_3]\backslash\!\backslash a \;\| \cdots \| \;[C_n]\backslash\!\backslash a$$

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:

$$([C_1]\backslash\!\backslash\{a, \overline{b}\}) \;\|\; ([C_2]\backslash\!\backslash\{\overline{a}, b\})$$

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

$$\big(([C]\|[C'])\backslash\!\backslash a\big) \;\|\; [D]$$

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

$$P \;::=\; [C] \;\;|\;\; P\|P \;\;|\;\; P\backslash\!\backslash L$$

where $C \in \mathcal{P}_{conSt}$ and $L \subseteq \mathcal{A}$.

We use P, P', $\cdots$, Q, Q', $\cdots$ to range over terms representing contract compositions, also called systems. The set of "initial" systems, i.e. systems in which only initial contracts $C \in \mathcal{P}_{con}$ occur, is denoted by $\mathcal{P}_{sys}$.

In the following we will sometimes omit parenthesis"[ ]" when writing contract compositions and we will call *system* a composition of contracts.

The operational semantics of systems is defined by the rules in Table 2 (plus the omitted symmetric rules). In the following we will use $\mathcal{P}_{sysSt}$ to denote the set of system states, defined as the terms that are reachable by a (possibly empty) sequence of transitions from "initial" systems in $\mathcal{P}_{sys}$.

Note that, due to the absence of internal communication of actions of $\mathcal{A}$ inside contracts, when we apply external restriction directly to a contract C, i.e. we consider $[C]\backslash\!\backslash L$ for some $L \subseteq \mathcal{A}$, we obtain a transition system isomorphic to that of the contract $C\{\mathbf{0}/\beta | \beta \in L\} \setminus \mathcal{N}_{con}$ or, equivalently, to that of $[C\{\mathbf{0}/\beta | \beta \in L\}]$, where $C\{\mathbf{0}/\beta | \beta \in L\}$ represents the syntactical substitution of $\mathbf{0}$ for every occurrence of any subterm β such that $\beta \in L$.

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

$$\frac{C \xrightarrow{\lambda} C' \quad \lambda \notin \mathcal{A}_{con}}{[C] \xrightarrow{\lambda} [C']} \qquad \frac{P \xrightarrow{\lambda} P' \quad \lambda \neq \surd}{P \| Q \xrightarrow{\lambda} P' \| Q}$$

$$\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P \| Q \xrightarrow{\tau} P' \| Q'} \qquad \frac{P \xrightarrow{\surd} P' \quad Q \xrightarrow{\surd} Q'}{P \| Q \xrightarrow{\surd} P' \| Q'} \qquad \frac{P \xrightarrow{\lambda} P' \quad \lambda \notin L}{P \backslash\backslash L \xrightarrow{\lambda} P' \backslash\backslash L}$$

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

the system is both deadlock and, under the fairness assumption [2], 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 2.3. (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 $C_1 \| C_2$ with

$$\begin{array}{ll} C_1 = a + b & C_2 = \overline{a} + \overline{b} \\ C_1 = a; b & C_2 = \overline{a}; \overline{b} \\ C_1 = a + b + c & C_2 = \overline{a} + \overline{b} \\ C_1 = (a; b) + (b; a) & C_2 = \overline{a} \,|\, \overline{b} \\ C_1 = (a; \overline{b})^* & C_2 = \overline{a}; (b; \overline{a})^*; b \end{array}$$

As an example of contract composition which is not correct we can consider $[\overline{a}; b] \| [a]$ in which the first service deadlocks after executing the first output action (thus successful completion cannot be reached). Another interesting example is $[(\overline{a}; b)^*] \| [a; (\overline{b}; a)^*]$, 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: $[(\overline{a}; b)^* + (\overline{c}; d)] \| [(a; (\overline{b}; a)^*) + c]$.

The design choice of requiring livelock freedom under the fairness assumption is related to considering: (i) unsatisfactory to just require deadlock freedom (accepting too many systems as correct ones as, e.g., the system above $[(\overline{a}; b)^*] \| [a; (\overline{b}; a)^*]$): 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 (ii) too demanding to require livelock freedom without the fairness assumption (discarding too many systems that intutively 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).

[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.

Example 2.1. (Travel Agency)
As an example of contract composition we consider the specification of a system composed of a client that sends a request to a travel agency. Upon reception of a client's request, the travel agency sends two parallel requests to the reservation services of a hotel and an airplane company. After having collected the replies from the reservation services, the travel agency either confirms or cancels the client's request.

$$
\begin{array}{l}
[\overline{Reservation}; (Confirmation + Cancellation)] \parallel \\
[Reservation; (\overline{ReserveFlight}; (ConfirmFlight; \overline{okFlight_*} + CancelFlight; \overline{koFlight_*}) \mid \\
\qquad \overline{ReserveRoom}; (ConfirmRoom; \overline{okRoom_*} + CancelRoom; \overline{koRoom_*}) \mid \\
\qquad (okFlight_*; (okRoom_*; \overline{Confirmation} + koRoom_*; \overline{Cancellation}) + \\
\qquad koFlight_*; (okRoom_*; \overline{Cancellation} + koRoom_*; \overline{Cancellation})))] \parallel \\
[ReserveFlight; (\overline{ConfirmFlight} + \overline{CancelFlight})] \parallel \\
[ReserveRoom; (\overline{ConfirmRoom} + \overline{CancelRoom})]
\end{array}
$$

The contracts of the client, of the hotel reservation service, and of the airplane company are simple. The travel agency is more complex because internal communication is used on the local names $okFlight$ and $okRoom$ in order to synchronize the answer to the client's request with the results received from the reservation services. Only if both reservations succeed, the travel agency sends a confirmation to the client. In all the other cases, a cancellation is sent.

It is not difficult to prove that the above contract composition is correct according to the Definition 2.3.

3. Contract Refinement

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 $C_1, \cdots, C_n$, we can replace each contract C_i with 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 take under consideration 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 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 initial contracts.

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

$$C_1 = a + b \qquad C_2 = \overline{a} + \overline{b}$$

We can replace C_1 with $C_1' = a + b + c$ or C_2 with $C_2' = \overline{a}$ 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

$$D_1 = a + b + c \qquad D_2 = \overline{a} + \overline{b}$$

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

$$D_1 = a + b + c \qquad D_2' = \overline{a} + \overline{b} + \overline{c}$$

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 $\overline{c}$.

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 $D_1' \not\leq_{\mathcal{N},\mathcal{N}} D_1$.

3.1. Subcontract pre-orders as correctness preserving refinements

We first define two auxiliary functions that extract from contracts and systems the set of names used in input and output actions, respectively.

Definition 3.1. (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. ouput) actions of C. Formally, we define $I(C)$ as follows:

$$I(\mathbf{0}) = I(\mathbf{1}) = I(\tau) = I(a_*) = I(\overline{a}_*) = I(\overline{a}) = \emptyset \qquad I(a) = \{a\}$$
$$I(C;C') = I(C+C') = I(C|C') = I(C) \cup I(C') \qquad I(C \setminus M) = I(C^*) = I(C)$$

and $O(C)$ as follows:

$$O(\mathbf{0}) = O(\mathbf{1}) = O(\tau) = O(a_*) = O(\overline{a}_*) = O(a) = \emptyset \qquad O(\overline{a}) = \{a\}$$
$$O(C;C') = O(C+C') = O(C|C') = O(C) \cup O(C') \qquad O(C \setminus M) = O(C^*) = O(C)$$

Note that the set M in $C \setminus M$ does not influence $I(C \setminus M)$ and $O(C \setminus M)$ because it contains only local names outside $\mathcal{N}$. Given the system P, we define $I(P)$ (resp. $O(P)$) as the subset of $\mathcal{N}$ of the potential input (resp. ouput) actions of P. Formally, we define $I(P)$ as follows:

$$I([C]) = I(C) \qquad I(P \| P') = I(P) \cup I(P') \qquad I(P \backslash\backslash L) = I(P) - L$$

and $O(P)$ as follows:

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

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 $C_1 \backslash\backslash I_1 \cup \overline{O}_1 \parallel \ldots \parallel C_n \backslash\backslash I_n \cup \overline{O}_n$ composed of the contracts $C_1, \cdots, C_n$ following a particular name discipline: the names in I_i (resp. O_i) cannot be used in input (resp. output) actions by the contract C_i. This discipline is guaranteed restricting each contract C_i on the set of actions $I_i \cup \overline{O_i}$. In this particular system, we want to be able to substitute each of the contract C_i with any contract C'_i such that $C'_i \leq_{I,O} C_i$ where I and O comprise the possible input and output actions that can be executed by the other contracts C_j with $j \neq i$. This last condition can be ensured imposing that

$$\big(\bigcup_{j \neq i} I(C_j) - I_j\big) - O_i \subseteq I \quad \wedge \quad \big(\bigcup_{j \neq i} O(C_j) - O_j\big) - I_i \subseteq O$$

This kind of formula is considered in the subsequent definition that formalizes the notion of subcontract pre-order family.

Definition 3.2. (Subcontract pre-order family) A family $\{\leq_{I,O} |\ I, O \subseteq \mathcal{N}\}$ of pre-orders over $\mathcal{P}_{con}$ is a subcontract pre-order family if, for any $n \geq 1$, contracts $C_1, \ldots, C_n \in \mathcal{P}_{con}$ and $C'_1, \ldots, C'_n \in \mathcal{P}_{con}$ and input and output names $I_1, \ldots, I_n \subseteq \mathcal{N}$ and $O_1, \ldots, O_n \subseteq \mathcal{N}$, we have

$$\begin{array}{l} (C_1 \backslash\backslash I_1 \cup \overline{O}_1 \parallel \ldots \parallel C_n \backslash\backslash I_n \cup \overline{O}_n)\downarrow \ \wedge \\ \forall i.\, C'_i \leq_{I'_i,O'_i} C_i \ \wedge \ (\bigcup_{j \neq i} I(C_j) - I_j) - O_i \subseteq I'_i \ \wedge \ (\bigcup_{j \neq i} O(C_j) - O_j) - I_i \subseteq O'_i \\ \qquad \Rightarrow \quad (C'_1 \backslash\backslash I_1 \cup \overline{O}_1 \parallel \ldots \parallel C'_n \backslash\backslash I_n \cup \overline{O}_n)\downarrow \end{array}$$

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 a mixed choice $a + \overline{b}$ in which the possible $\overline{b}$ is no longer executable after the input a) it is easy to prove that there exists no maximal subcontract pre-order family. Suppose that the semantics of outputs "$\overline{a}$" is defined by means of the following operational rule that is similar to the one used for inputs "a":

$$\overline{a} \xrightarrow{\overline{a}} \mathbf{1}$$

Now consider, e.g., the trivially correct system $C_1 \| C_2$ with $C_1 = a$ and $C_2 = \overline{a}$; we could have two subcontract pre-order families $\leq^1$ and $\leq^2$ such that

$$a + c.\mathbf{0} \leq^1_{\mathcal{N}-c,\mathcal{N}-c} a \ \text{ and } \ \overline{a} + c.\mathbf{0} \leq^1_{\mathcal{N}-c,\mathcal{N}-c} \overline{a}$$

and

$$a + \overline{c}.\mathbf{0} \leq^2_{\mathcal{N}-c,\mathcal{N}-c} a \ \text{ and } \ \overline{a} + \overline{c}.\mathbf{0} \leq^2_{\mathcal{N}-c,\mathcal{N}-c} \overline{a}$$

but no subcontract pre-order family $\leq$ could have

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

because if we refine C_1 with $a+c.\mathbf{0}$ and C_2 with $\overline{a}+\overline{c}.\mathbf{0}$ we achieve the incorrect system $a+c.\mathbf{0}\|\overline{a}+\overline{c}.\mathbf{0}$ that can deadlock after synchronization on channel c.

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.

3.2. Input-output subcontract relation as the maximal subcontract pre-order

We will show that 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 relation is coarser than the subcontract pre-order.

Intuitively a family of pre-orders $\{\leq_{I,O} |\ I,O \subseteq \mathcal{N}\ \}$ is a singular subcontract pre-order family whenever the correctness of systems is preserved by refining just one of the contracts. More precisely, for any $n \geq 1$, $C_1,\ldots,C_n \in \mathcal{P}_{con}$, $I_1,\ldots,I_n \subseteq \mathcal{N}$, $O_1,\ldots,O_n \subseteq \mathcal{N}$, $1 \leq i \leq n$ and $C'_i \in \mathcal{P}_{con}$ we require

$$(C_1 \backslash\backslash I_1 \cup \overline{O}_1 \parallel \ldots \parallel C_i \backslash\backslash I_i \cup \overline{O}_i \parallel \ldots \parallel C_n \backslash\backslash I_n \cup \overline{O}_n)\downarrow \ \wedge$$
$$C'_i \leq_{I,O} C_i \ \wedge\ (\textstyle\bigcup_{j\neq i} I(C_j) - I_j) - O_i \subseteq I \ \wedge\ (\bigcup_{j\neq i} O(C_j) - O_j) - I_i \subseteq O$$
$$\Rightarrow \quad (C_1 \backslash\backslash I_1 \cup \overline{O}_1 \parallel \ldots \parallel C'_i \backslash\backslash I_i \cup \overline{O}_i \parallel \ldots \| C_n \backslash\backslash I_n \cup \overline{O}_n)\downarrow$$

By exploiting commutativity and associativity of parallel composition, and the fact that the internal behavior of $C_1 \backslash\backslash I_1 \cup \overline{O}_1 \ \parallel \ldots \parallel\ C_n \backslash\backslash I_n \cup \overline{O}_n$ is the same as that of $C_1 \| ((C_2\{\mathbf{0}/\beta | \beta \in I_2 \cup \overline{O}_2\} \| \ldots \| C_n\{\mathbf{0}/\beta | \beta \in I_n \cup \overline{O}_n\}) \backslash\backslash O_1 \cup \overline{I}_1)$ we can group the contracts which are not being refined and denote them with a generic term P taken from $\mathcal{P}_{conpar}$, the set of the systems of the form $(C_1 \| \ldots \| C_n) \backslash\backslash I \cup \overline{O}$, with $C_i \in \mathcal{P}_{con}$ for all $i \in \{1,\ldots,n\}$ and $I,O \subseteq \mathcal{N}$. Moreover we note that, given $P = (C_1 \| \ldots \| C_n) \backslash\backslash I \cup \overline{O} \in \mathcal{P}_{conpar}$, we have $I(P) = (\bigcup_{1\leq i\leq n} I([C_i])) - I$ and $O(P) = (\bigcup_{1\leq i\leq n} O([C_i])) - O$.

Definition 3.3. (Singular subcontract pre-order family) A family of pre-orders $\{\leq_{I,O} |\ I,O \subseteq \mathcal{N}\ \}$ is a singular subcontract pre-order family if, for any $C, C' \in \mathcal{P}_{con}$, $P \in \mathcal{P}_{conpar}$ we have

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

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 other unchanged.

Proposition 3.1. If a family of pre-orders $\{\leq_{I,O} |\ I, O \subseteq \mathcal{N}\}$ is a subcontract pre-order family then it is also a singular subcontract pre-order family.

Proof:
Suppose that $\{\leq_{I,O} |\ I, O \subseteq \mathcal{N}\}$ is a subcontract pre-order family. Consider $n \geq 1$, $P \in \mathcal{P}_{conpar}$, $C, C' \in \mathcal{P}_{con}$. From $(C \| P)\downarrow$ and $C' \leq_{I,O} C$, where $I(P) \subseteq I$ and $O(P) \subseteq O$, we can derive $(C' \| P)\downarrow$ by just taking in the definition of subcontract pre-order family, $C_1 = C$, $C'_1 = C'$, $C_2 \ldots C_n$ and I_1 and O_1 to be such that $P = (C_2 \| \ldots \| C_n) \backslash\backslash O_1 \cup \overline{I}_1$; $I_2 \ldots I_n$ and $O_2 \ldots O_n$ to be the emptyset; and finally C'_i to be C_i for every $i \geq 2$ (since $\leq_{I,O}$ are pre-orders we have $C \leq_{I,O} C$ for every I, O and C). □

In the following we let $\mathcal{P}_{conpar,I,O}$ denote the subset of processes of $\mathcal{P}_{conpar}$ such that $I(P) \subseteq I$ and $O(P) \subseteq O$. We start to use this notation in the definition of the input-output subcontract relation.

Definition 3.4. (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

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

The difference between the Definition 3.3 and the Definition 3.4 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 I-O 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 I-O 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 I-O subcontract relation (Definition 3.4), but we have presented both definitions because this simplify the proof of the following Theorem (stating an important property about the I-O subcontract relation) that, indeed, is a simple corollary of the Proposition 3.1 in which we made use of the Definition 3.3.

Theorem 3.1. Given a subcontract pre-order family $\{\leq_{I,O} |\ I, O \subseteq \mathcal{N}\}$, we have that it is *included* in $\{\preceq_{I,O} |\ I, O \subseteq \mathcal{N}\}$, that is

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

Proof:
From Proposition 3.1 we know that each subcontract pre-order family $\{\leq_{I,O} |\ I, O \subseteq \mathcal{N}\}$ is also a singular subcontract pre-order family. The thesis directly follows from the definition of subcontract relation, as it easy to see that $\preceq_{I,O}$ coincides with the union of all singular subcontract pre-orders $\leq_{I,O}$. □

In light of this last Theorem, the existence of the maximal subcontract pre-order family can be proved simply showing that $\{\preceq_{I,O} |\ I, O \subseteq \mathcal{N}\}$ is itself a subcontract pre-order family (thus it is the maximum among all subcontract pre-order family). The proof of this result (Theorem 3.2) 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 3.2. Let $C, C' \in \mathcal{P}_{con}$ be two contracts, $I, I' \subseteq \mathcal{N}$ be two sets of input channel names such that $I' \subseteq I$ and $O, O' \subseteq \mathcal{N}$ be two sets of output channel names such that $O' \subseteq O$. We have:

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

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

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 3.2.

Proposition 3.3. Let $C, C' \in \mathcal{P}_{con}$ be contracts and $I, O \subseteq \mathcal{N}$ be sets of input and output names. We have

$$C' \preceq_{I,O} C \quad \Rightarrow \quad C' \backslash\backslash (I(C') - I(C)) \preceq_{I,O} C$$
$$C' \preceq_{I,O} C \quad \Rightarrow \quad C' \backslash\backslash \overline{(O(C') - O(C))} \preceq_{I,O} C$$

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 $P \in \mathcal{P}_{conpar,I,O}$ such that $(C \| P)\downarrow$, we will show that $(C' \backslash\backslash \overline{(O(C') - O(C))} \parallel P)\downarrow$. We first observe that $(C \parallel P \backslash\backslash (O(C') - O(C)))\downarrow$. Since $C' \preceq_{I,O} C$, we derive $(C' \parallel P \backslash\backslash (O(C') - O(C)))\downarrow$. As a consequence $(C' \backslash\backslash \overline{(O(C') - O(C))} \parallel P \backslash\backslash (O(C') - O(C)))\downarrow$. We can conclude $(C' \backslash\backslash \overline{(O(C') - O(C))} \parallel P)\downarrow$. □

All the results discussed so far do not depend on the output persistence property. The first relevant result depending on this peculiarity 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 3.4. 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 $P \in \mathcal{P}_{conpar,I,O}$ such that $(C \| P)\downarrow$,

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

Proof:
We proceed by contradiction. Suppose that there exist C'_{der}, P_{der} such that $(C' \| P) \xrightarrow{\tau}{}^{*} (C'_{der} \| P_{der})$ and $C'_{der} \xrightarrow{\overline{a}}$ 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 $(C'_{der2} \| P_{der2})$ is traversed, such that $C'_{der2} \xrightarrow{\overline{a}}$ for some $a \in O(C') - O(C)$. This implies that the same path must be performable by $(C' \backslash\backslash \overline{(O(C') - O(C))} \parallel P)$, thus reaching the state $(C'_{der} \backslash\backslash \overline{(O(C') - O(C))} \parallel P_{der})$. However, since in the state C'_{der} of contract C' we have $C'_{der} \xrightarrow{\overline{a}}$ for some $a \in O(C') - O(C)$ and the execution of $\overline{a}$ 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 $(C' \backslash\backslash \overline{(O(C') - O(C))} \parallel P)\not\downarrow$ and (due to Proposition 3.3) we reached a contradiction.
□

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 3.5. 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 $C' \in \mathcal{P}_{con}$,

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

Proof:
Let us suppose $C' \preceq_{I',O} C$ (the opposite direction is symmetric). Given any $P \in \mathcal{P}_{conpar,I,O}$ such that $(C \| P) \downarrow$, we will show that $(C' \| P) \downarrow$. We first observe that $(C \parallel P \backslash\backslash (I - O(C))) \downarrow$. Since $C' \preceq_{I',O} C$ and $O(C) \subseteq I'$, we derive $(C' \parallel P \backslash\backslash (I - O(C))) \downarrow$. Due to Proposition 3.4 we have that $(C' \parallel P \backslash\backslash (I - O(C)))$ 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 $(C' \parallel P) \downarrow$. □

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 $C' \not\preceq_{I,O'} C$. As an example, you can consider $\overline{a} + b \preceq_{\mathcal{N},\mathcal{N}-b} \overline{a}$. 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 $\overline{a} + b \not\preceq_{\mathcal{N},\mathcal{N}} \overline{a}$ because

$$[\overline{a}] \parallel [a; b] \parallel [\overline{b}]$$

is a correct composition while

$$[\overline{a} + b] \parallel [a; b] \parallel [\overline{b}]$$

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 the main result of this paper, i.e., that the input-output subcontract relation defined in the Definition 3.4 is also a subcontract pre-order family.

Theorem 3.2. The family of pre-orders $\{\preceq_{I,O} |\ I, O \subseteq \mathcal{N}\}$ is a subcontract pre-order family.

Proof:
Consider $n \geq 1$, $C_1, \ldots, C_n \in \mathcal{P}_{con}$, $C'_1, \ldots, C'_n \in \mathcal{P}_{con}$, $I_1, \ldots, I_n \subseteq \mathcal{N}$ and $O_1, \ldots, O_n \subseteq \mathcal{N}$. For any i we let $P_i = C_i \backslash\backslash I_i \cup \overline{O}_i$ and $P'_i = C'_i \backslash\backslash I_i \cup \overline{O}_i$. If $(P_1 \| \ldots \| P_n) \downarrow$ and for all i we have that $C'_i \preceq_{I'_i,O'_i} C_i$, with I'_i and O'_i satisfying the constraint on names as specified in Definition 3.2, we can derive $(P'_1 \| \ldots \| P'_n) \downarrow$ as follows. For every i from 1 to n we show that

$$(P'_1 \backslash\backslash \overline{(O(C'_1) - O(C_1))} \| \ldots \| P_i \| \ldots \| P'_n \backslash\backslash \overline{(O(C'_n) - O(C_n))}) \downarrow$$

by multiply applying the definition of singular subcontract pre-order family to any C_j with $j \neq i$. For instance if i is 1, from $(P_1 \| \ldots \| P_n) \downarrow$ we derive

$$(P_1 \| P'_2 \backslash\backslash \overline{(O(C'_2) - O(C_2))} \| P_3 \| \ldots \| P_n) \downarrow$$

by applying the definition of singular subcontract pre-order to refine C_2 and by using Proposition 3.3. We then use this intermediate result to re-apply the definition of singular subcontract pre-order family for refining C_3 and we derive

$$(P_1 \| P_2' \backslash\backslash \overline{(O(P_2') - O(P_2))} \| P_3' \backslash\backslash \overline{(O(C_3') - O(C_3))} \| P_4 \| \ldots \| P_n) \downarrow$$

We proceed in this way until we yield

$$(P_1 \| P_2' \backslash\backslash \overline{(O(C_2') - O(C_2))} \| \ldots \| P_n' \backslash\backslash \overline{(O(C_n') - O(C_n))}) \downarrow$$

For $i \in \{2 \ldots n\}$ we proceed in a similar way to obtain

$$(P_1' \backslash\backslash \overline{(O(C_1') - O(C_1))} \| \ldots \| P_i \| \ldots \| P_n' \backslash\backslash \overline{(O(C_n') - O(C_n))}) \downarrow$$

We conclude the proof as follows. For any i, since $C_i' \preceq_{I,O} C_i$, by Proposition 3.4 we have that $(P_1' \backslash\backslash \overline{(O(C_1') - O(C_1))} \| \ldots \| P_i' \| \ldots \| P_n' \backslash\backslash \overline{(O(C_n') - O(C_n))})$ can never reach by τ transitions a state where outputs in $O(C_1') - O(C_1)$ are executable by the derivative $C_{i,der}'$ of C_i' that is syntactically included in the derivative $P_{i,der}'$ of P_i'. If now we consider the behavior of

$$(P_1' \backslash\backslash \overline{(O(C_1') - O(C_1))} \| \ldots \| P_n' \backslash\backslash \overline{(O(C_n') - O(C_n))})$$

we derive that, for any i, we cannot reach by τ transitions a state

$$(P_{1,der}' \backslash\backslash \overline{(O(C_1') - O(C_1))} \| \ldots \| (P_{i,der}' \backslash\backslash \overline{(O(C_i') - O(C_i))} \| \ldots \| P_{n,der}' \backslash\backslash \overline{(O(C_n') - O(C_n))})$$

where $C_{i,der}'$ (the derivative of C_i' syntactically included in $P_{i,der}'$) can execute outputs in $O(C_i') - O(C_i)$. Hence the presence of the restriction operators does not affect the internal behavior of

$$(P_1' \backslash\backslash \overline{(O(C_1') - O(C_1))} \| \ldots \| P_n' \backslash\backslash \overline{(O(C_n') - O(C_n))})$$

with respect to $(P_1' \| \ldots \| P_n')$. Therefore, we can finally derive $(P_1' \| \ldots \| P_n') \downarrow$ from

$$(P_1' \backslash\backslash \overline{(O(C_1') - O(C_1))} \| \ldots \| P_n' \backslash\backslash \overline{(O(C_n') - O(C_n))}) \downarrow$$

that is obtained by further applying the definition of singular subcontract pre-order to refine C_i in any of the i-indexed statement in the first part of the proof. □

In Theorem 3.1 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.

3.3. Subcontract relation

In the previous subsection we have introduced the input-output subcontract relation $\preceq_{I,O}$, and we have proved that the set of all input-output subcontract relations is the maximal subcontract pre-order family. Moreover, the Proposition 3.5 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 $\preceq_O$, 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 $\mathcal{P}_{conpar,O}$ to denote the set of processes $\mathcal{P}_{conpar,\mathcal{N},\mathcal{O}}$.

Definition 3.5. (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

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

It is easy to see that $\preceq_O = \preceq_{\mathcal{N},\mathcal{O}}$.

In order to prove that one contract C' is a subcontract of C, the Definition 3.5 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* [23]. The main difference of should-testing with respect to the standard must-testing [15] 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 3.3.

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

Lemma 3.1. Let C, C' be two contracts and $O \subseteq \mathcal{N}$ be a set of output names. We have $C' \preceq_O C$ iff

$$\forall P \in \mathcal{P}_{conpar}. \quad (C \backslash\!\backslash \mathcal{N} - O \,\|\, P)\downarrow \quad \Rightarrow \quad (C' \backslash\!\backslash \mathcal{N} - O \,\|\, P)\downarrow$$

Proof:
Given $P \in \mathcal{P}_{conpar}$, we have $(C \backslash\!\backslash \mathcal{N} - O \,\|\, P)\downarrow \iff (C \backslash\!\backslash \mathcal{N} - O \,\|\, P \backslash\!\backslash \overline{\mathcal{N} - O})\downarrow \iff (C \,\|\, P \backslash\!\backslash \overline{\mathcal{N} - O})\downarrow$ and $(C' \backslash\!\backslash \mathcal{N} - O \,\|\, P)\downarrow \iff (C' \backslash\!\backslash \mathcal{N} - O \,\|\, P \backslash\!\backslash \overline{\mathcal{N} - O})\downarrow \iff (C' \,\|\, P \backslash\!\backslash \overline{\mathcal{N} - O})\downarrow$. In the particular case of $P \in \mathcal{P}_{conpar,O}$ we have that $P \backslash\!\backslash \overline{\mathcal{N} - O}$ is isomorphic to P. □

In the following we denote with $\preceq_{test}$ the *should-testing* pre-order defined in [23] where we consider the set of actions Λ used by terms as being $\Lambda = \mathcal{A} \cup \{\surd\}$ (i.e. we consider non-contract input and output actions and $\surd$: the latter is included in the set of actions of terms being tested as any other action). We denote here with $\surd'$ the special action for the success of the test (denoted by $\surd$ in [23]). Similarly as in [23] we use Λ_τ to denote $\Lambda \cup \{\tau\}$.

In order to resort to the theory defined in [23], we define a normal form representation with terms of the process algebra considered in [23] of the finite labeled transition system (LTS) of a system P. 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 $[\![P]\!]$ of a system P is defined as being the LTS $[\![P]\!] = (S, \Lambda_\tau, \longrightarrow, P)$, 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 2.1 and 2, i.e. such that $P \stackrel{w}{\longrightarrow} P'$ 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 $\mathcal{N}F(P)$) is derived from its semantics $[\![P]\!] = (S, \Lambda_\tau, \longrightarrow, P)$ as follows, by using the operator $rec_X \theta$ (defined in [23]) that represents the value of X in the solution of the minimum fixpoint of the finite set of equations θ,

$$\begin{aligned} \mathcal{N}F(P) &= rec_{X_P}\theta \qquad \text{where } \theta \text{ is the set of } S\text{-indexed equations} \\ X_{P'} &= \textstyle\sum_{(\lambda,P''):P' \stackrel{\lambda}{\longrightarrow} P''} \lambda; X_{P''} \end{aligned}$$

where we assume empty sums to be equal to $\mathbf{0}$, i.e. if there are no outgoing transitions from $X_{P'}$, we have $X_{P'} = \mathbf{0}$.

According to the definitions in [23], the semantics $[\![\mathcal{NF}(P)]\!]$ of the normal form $\mathcal{NF}(P) \equiv rec_{X_P}\theta$ is, as expected, the labeled transition system $[\![\mathcal{NF}(P)]\!] = (S', \Lambda_\tau, \longrightarrow', \mathcal{NF}(P))$, where:

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

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

Theorem 3.3. Let C, C' be two contracts and $O \subseteq \mathcal{N}$ be a set of output names. We have

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

Before reporting the proof of this theorem, we first introduce some technical machinery.

In order to build a test for the transformation $\mathcal{NF}(C \backslash\!\backslash (\mathcal{N} - O))$ 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 $\surd'$ 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 [23].

We first consider $\mathcal{NF}'(P) \equiv \mathcal{NF}(P)\{\surd; \surd'; X_{P'}/\surd; X_{P'} \mid P' \in \mathcal{P}_{sysSt}\}$, 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 $\surd; X_{P'}$ occurring in θ, for any P', with the subterm $\surd; \surd'; X_{P'}$. The LTS $[\![\mathcal{NF}'(P)]\!] = (S'', \Lambda_\tau, \longrightarrow'', \mathcal{NF}'(P))$ turns out to be, according to the definitions in [23], as follows

- $S'' = \{\mathcal{NF}'(P') \equiv rec_{X_{P'}}\theta' \mid P' \in S\} \cup \{\mathcal{NF}'_{\surd}(P'') \equiv \surd'.rec_{X_{P''}}\theta' \mid \exists P' \in S : P' \stackrel{\surd}{\longrightarrow} P''\}$
- $\longrightarrow'' = \{(\mathcal{NF}'(P'), \lambda, \mathcal{NF}'(P'')) \mid P' \stackrel{\lambda}{\longrightarrow} P'' \wedge \lambda \neq \surd\}$
  $\cup \{(\mathcal{NF}'(P'), \surd, \mathcal{NF}'_{\surd}(P'')) \mid P' \stackrel{\surd}{\longrightarrow} P''\}$
  $\cup \{(\mathcal{NF}'_{\surd}(P''), \surd', \mathcal{NF}'(P'')) \mid \mathcal{NF}'_{\surd}(P'') \in S''\}$

where we assume $(S, \Lambda_\tau, \longrightarrow, P)$ to denote the LTS $[\![P]\!]$.

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 $\overline{a}$ and every $\overline{a}$ occurring in θ', for any $a \in \mathcal{N}$, into a. The LTS $[\![\overline{\mathcal{NF}'(P)}]\!] = (S''', \Lambda_\tau, \longrightarrow''', \overline{\mathcal{NF}'(P)})$ turns out to be a transformation of $[\![\mathcal{NF}'(P)]\!]$ 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 $\mathcal{NF}'_{\surd}(P')$ is turned into $\overline{\mathcal{NF}'_{\surd}(P')}$) and whose transition labels are transformed by inverting input/output actions as described above.

We now introduce mapping of traces of $[\![[C]]\!]$ into $[\![\mathcal{NF}(C)]\!]$ and mapping of traces of $[\![P]\!]$ into $[\![\mathcal{NF}'(P)]\!]$. First of all we define a n-length trace $tr \in Tr_n^{\mathcal{T}}$, 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 $s_{i-1} \xrightarrow{\lambda_i} s_i \forall 1 \leq i \leq n$. A n-length initial trace $tr \in ITr_n^{\mathcal{T}}$ is defined in the same way with the additional constraint that $s_0 = s_{init}$. We let $Tr^{\mathcal{T}}$ to stand for $\bigcup_{n \geq 0} Tr_n^{\mathcal{T}}$. In the following we will also denote a n-length trace tr simply by writing the sequence of its transitions, i.e. $tr = s_0 \xrightarrow{\lambda_1} s_1 \xrightarrow{\lambda_2} \ldots \xrightarrow{\lambda_{n-1}} s_{n-1} \xrightarrow{\lambda_n} s_n$. 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 $tr' \frown tr''$ defined as the trace $tr \in Tr_{n+m}^{\mathcal{T}}$ with $s_i = s'_i \forall 0 \leq i \leq n$, $\lambda_i = \lambda'_i \forall 1 \leq i \leq n$, $s_{n+i} = s''_i \forall 1 \leq i \leq m$ and $\lambda_{n+i} = \lambda''_i \forall 1 \leq i \leq m$. 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 $s'_i = s_i \forall 0 \leq i \leq n-i$ and $\lambda'_i = \lambda_i \forall 1 \leq i \leq n-i$. 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 $\lambda_n = \tau$, $vis(tr) = vis(less(tr)) \frown \lambda_n$ otherwise (where we use $w' \frown w''$ to denote string concatenation).

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

Let us consider a transition $s \xrightarrow{\lambda} s'$ of $[\![P]\!]$. We can take $tr = s \xrightarrow{\lambda} s'$ with $tr \in Tr_1^{[\![P]\!]}$. We define $map_{\surd}(tr)$ as follows. If $\lambda \neq \surd$ then $map_{\surd}(tr) = \mathcal{NF}'(s) \xrightarrow{\lambda} \mathcal{NF}'(s')$. Otherwise, if $\lambda = \surd$ then $map_{\surd}(tr) = \mathcal{NF}'(s) \xrightarrow{\surd} \mathcal{NF}'_{\surd}(s')$. We then define the mapping $map_{\surd}(tr)$ of a whatever transition $tr = (s, \lambda) \in Tr_n^{[\![P]\!]}$ (tr may include $\surd$ only as the final transition because target states of $\surd$ transitions have no outgoing transitions in the semantics of systems) to be the transition $tr' = (s', \lambda') \in Tr_n^{[\![\mathcal{NF}(P)]\!]}$ with $s'_0 = \mathcal{NF}'(s_0)$ and, in the case $n \geq 1$, $s'_n = \mathcal{NF}'(s_n)$ if $\lambda_n \neq \surd$, $s'_n = \mathcal{NF}'_{\surd}(s_n)$ otherwise, achieved by induction on $n \geq 0$ as follows. If $n = 0$ then $map_{\surd}(tr)$ is the trace $tr' \in Tr_0^{[\![\mathcal{NF}'(P)]\!]}$ such that $s'_0 = \mathcal{NF}'(s_0)$. If $n \geq 1$ then $map_{\surd}(tr) = map_{\surd}(less(tr)) \frown map_{\surd}(s_{n-1} \xrightarrow{\lambda_n} s_n)$. It is immediate to verify that for any $tr \in Tr^{[\![P]\!]}$, $vis(map_{\surd}(tr)) = vis(tr)$. Moreover we have that $map_{\surd} : Tr^{[\![P]\!]} \rightarrow Tr^{[\![\mathcal{NF}'(P)]\!]}$ 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 $s \in [\![P]\!]$ and do not include $\surd'$ transitions.

We finally define some other auxiliary functions that will be used in the proof. Given a trace $tr \in Tr^{[\![\mathcal{NF}'(P)]\!]}$ or $tr \in Tr^{[\![\overline{\mathcal{NF}'(P)}]\!]}$, we define $\overline{tr}$ to be $tr' = (s', \lambda')$ defined as: $s'_i = \overline{s_i} \forall i$ and $\lambda'_i = \overline{\lambda_i} \forall i$ (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 $\surd$ labels). Notice that $vis(\overline{tr}) = \overline{vis(tr)}$ denoting the application of the overbar to any label occurring in the sequence of visible labels.

Proof of Theorem 3.3:

According to the definition of should-testing of [23], since

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

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

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

where $\|_\Lambda$ is the CSP parallel operator: in $R\|_\Lambda R'$ transitions of R and R' with the same label λ (with $\lambda \neq \tau, \surd'$) are required to synchronize and yield a transition with label λ.

Let us now consider $P \in P_{conpar}$ with $(C\backslash\backslash\mathcal{N}-O\|P)\downarrow$. In the following we will provide a first subproof that this implies $\mathcal{NF}(C\backslash\backslash(\mathcal{N}-O))$ **shd** $\overline{\mathcal{NF}'(P)}$. Since $\mathcal{NF}(C'\backslash\backslash(\mathcal{N}-O)) \preceq_{test} \mathcal{NF}(C\backslash\backslash(\mathcal{N}-O))$, from this we can derive $\mathcal{NF}(C'\backslash\backslash(\mathcal{N}-O))$ **shd** $\overline{\mathcal{NF}'(P)}$. In the following we will provide a second subproof that this implies $(C'\backslash\backslash\mathcal{N}-O\|P)\downarrow$. The thesis, then, directly follows from Lemma 3.1.

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

We consider the trace $tr = (s,\lambda) \in ITr_r^{[\![\mathcal{NF}(\tilde{C})\|_\Lambda \overline{\mathcal{NF}'(P)}]\!]}$ such that $\lambda_i \neq \surd' \forall i$. There exist $\dot{tr} \in ITr_n^{[\![\mathcal{NF}(\tilde{C})]\!]}$ and $\ddot{tr} \in ITr_m^{[\![\overline{\mathcal{NF}'(P)}]\!]}$, 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. $\ddot{s}_m \equiv \overline{\mathcal{NF}'_{\surd}(s)}$ for some $s \in [\![P]\!]$

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

We now develop the non-trivial case (1) for $\ddot{tr}$. Let us consider $\ddot{tr}' = map_{\surd}^{-1}(\overline{\ddot{tr}}) \in ITr_m^{[\![P]\!]}$. We have $\ddot{s}_m = \overline{\mathcal{NF}'(\ddot{s}'_m)}$. Let us also consider $\dot{tr}' = map^{-1}(\dot{tr}) \in ITr_n^{[\![[\tilde{C}]]\!]}$. We have $\dot{s}_n = \mathcal{NF}(\dot{s}'_n)$. Moreover, $vis(\ddot{tr}') = \overline{vis(\ddot{tr})} = \overline{vis(\dot{tr})} = \overline{vis(\dot{tr}')}$.

Therefore, there exists $tr' \in ITr_r^{[\![\tilde{C}\|P]\!]}$, with $\lambda'_i = \tau \forall i$, such that $s'_r = \dot{s}'_n\|\ddot{s}'_m$.

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

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

Therefore, there exists $tr''' \in Tr_{r''}^{[\![\mathcal{NF}(\tilde{C})\|_\Lambda \overline{\mathcal{NF}'(P)}]\!]}$. Such trace has initial state $s'''_0 = \mathcal{NF}(\dot{s}'_n)\|_\Lambda \overline{\mathcal{NF}'(\ddot{s}'_m)} = \dot{s}_n\|_\Lambda \ddot{s}_m = s_r$, and $s'''_{r''} = \dot{s}'''_{n''}\|_\Lambda \ddot{s}'''_{m''}$. Moreover, since, by def. of $\mathcal{NF}'(P)$, $\ddot{\lambda}'''_{m''} = \surd \Rightarrow \ddot{s}'''_{m''} \xrightarrow{\surd'}$, we have $s'''_{r''} \xrightarrow{\surd'}$.

Second subproof: $\mathcal{NF}(\tilde{C}')$ **shd** $\overline{\mathcal{NF}'(P)} \Rightarrow (\tilde{C}'\|P)\downarrow$, with $\tilde{C}' = C'\backslash\backslash(\mathcal{N}-O)$.

We consider the trace $tr = (s,\lambda) \in ITr_r^{[\![\tilde{C}'\|P]\!]}$ such that $\lambda_i = \tau \forall i$. 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 $vis(\dot{tr}) = \overline{vis(\ddot{tr})}$.
Let us now consider $\dot{tr}' = map(\dot{tr}) \in ITr_n^{[\![\mathcal{NF}(\tilde{C}')]\!]}$. We have $\dot{s}'_n = \mathcal{NF}(\dot{s}_n)$. Let us also consider $\ddot{tr}' = \overline{map_{\surd}(\ddot{tr})} \in ITr_m^{[\![\overline{\mathcal{NF}'(P)}]\!]}$. We have $\ddot{s}'_m = \overline{\mathcal{NF}'(\ddot{s}_m)}$ because $\ddot{tr}$ does not include a $\surd$ transition. Moreover, $vis(\dot{tr}') = vis(\dot{tr}) = \overline{vis(\ddot{tr})} = vis(\ddot{tr}')$. Therefore, there exists $tr' \in ITr_r^{[\![\mathcal{NF}(\tilde{C}')\|_\Lambda \overline{\mathcal{NF}'(P)}]\!]}$ with $s'_r = \dot{s}'_n \|_\Lambda \ddot{s}'_m = \mathcal{NF}(\dot{s}_n)\|_\Lambda \overline{\mathcal{NF}'(\ddot{s}_m)}$.
Since $\mathcal{NF}(\tilde{C}')$ **shd** $\overline{\mathcal{NF}'(P)}$ we have that there exists $tr'' \in Tr_{r''}^{[\![\mathcal{NF}(\tilde{C}')\|_\Lambda \overline{\mathcal{NF}'(P)}]\!]}$ with $s''_0 = s'_r$, such that $\lambda''_i \neq \surd' \forall 1 \leq i \leq r''-1$ and $\lambda''_{r''} = \surd'$.
There exist $\dot{tr}'' \in Tr_{n''}^{[\![\mathcal{NF}(\tilde{C}')]\!]}$ with $\dot{s}''_0 = \dot{s}'_n$ and $\ddot{tr}'' \in Tr_{m''}^{[\![\overline{\mathcal{NF}'(P)}]\!]}$, 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 $\ddot{\lambda}''_{m''} = \surd'$ 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) $\ddot{\lambda}''_{m''-1} = \surd$. Moreover we must have $vis(\dot{tr}'') = vis(less(\ddot{tr}''))$, hence in particular $\dot{\lambda}''_{n''} = \surd$: the $\surd$ transition must be the last one in such a trace (i.e. there are no τ transitions afterward) because target states of $\surd$ 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 $\dot{tr}''' = map^{-1}(\dot{tr}'') \in Tr_{n''}^{[\![[\tilde{C}']]\!]}$. We have $\dot{s}'''_0 = \dot{s}'_n = \mathcal{NF}(\dot{s}'''_0)$ and $\dot{\lambda}'''_{n''} = \surd$. Let us also consider $\ddot{tr}''' = map_{\surd}^{-1}(\overline{less(\ddot{tr}'')}) \in Tr_{m''-1}^{[\![P]\!]}$. We have $\ddot{s}'''_0 = \dot{s}'_m = \overline{\mathcal{NF}'(\ddot{s}'''_0)}$ and $\ddot{\lambda}'''_{m''-1} = \surd$. Moreover, $vis(\ddot{tr}''') = \overline{vis(less(\ddot{tr}''))} = \overline{vis(\dot{tr}'')} = \overline{vis(\dot{tr}''')}$.
Therefore, there exists $tr''' \in Tr_{r''}^{[\![\tilde{C}'\|P]\!]}$ such that $\lambda'''_i = \tau \forall 1 \leq i \leq r''-1$, with $s'''_0 = \dot{s}'''_0 \|_\Lambda \ddot{s}'''_0 = \dot{s}_n \|_\Lambda \ddot{s}_m = s_r$ and $\lambda'''_{r''} = \surd$. □

Note that the opposite implication

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

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 $(C\|P)\downarrow$ or $(C'\|P)\downarrow$), but obviously $\mathcal{NF}(C' \backslash\!\backslash (\mathcal{N}-O)) \preceq_{test} \mathcal{NF}(C \backslash\!\backslash (\mathcal{N}-O))$ (and $\mathcal{NF}(C \backslash\!\backslash (\mathcal{N}-O)) \preceq_{test} \mathcal{NF}(C' \backslash\!\backslash (\mathcal{N}-O))$) 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 $\mathcal{NF}(C' \backslash\!\backslash (\mathcal{N}-O)) \preceq_{test} \mathcal{NF}(C \backslash\!\backslash (\mathcal{N}-O))$ (and $\mathcal{NF}(C \backslash\!\backslash (\mathcal{N}-O)) \preceq_{test} \mathcal{NF}(C' \backslash\!\backslash (\mathcal{N}-O))$) does not hold for any O that includes $\{a, b\}$: this can be seen by considering the test $t = \surd' + b; \mathbf{0}$ ($t = \surd' + a; \mathbf{0}$).

Finally, we observe that the labeled transition system of each contract C is finite state as we consider the Kleene-star repetition operator and not general recursion. This implies that also $\mathcal{NF}(C \backslash\!\backslash \mathcal{N} - O)$ is finite state for any O. In [23] 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 3.3, represents a sound approach to prove also our subcontract relation.

Example 3.1. (Refining the Contracts in the Travel Agency Example)
As an example of contract refinements, we consider the contracts presented in the travel agency example introduced as Example 2.1. Exploiting the Theorem 3.3 it is easy to prove the following refinement that

takes under consideration a client with additional behavior, i.e., the possibility to resend the reservation request in case a reply $Retry$ is received.

$$\overline{Reservation};(Confirmation + Cancellation + Retry;\overline{Reservation})$$
$$\preceq_{\{ConfirmFlight,CancelFlight,ConfirmRoom,CancelRoom,ReserveFlight,ReserveRoom,Confirmation,Cancellation\}}$$
$$\overline{Reservation};(Confirmation + Cancellation)$$

Note that we put in the set of names subscript of $\preceq$ all those names on which the other contracts performs output actions.

As other examples of refinement, we consider an airplane reservation service that always replies positively to every request and a hotel reservation service that always replies negatively. Also in this case, using the Theorem 3.3, it is easy to prove that

$$ReserveFlight;\overline{ConfirmFlight}$$
$$\preceq_{\{Reservation,ConfirmRoom,CancelRoom,ReserveFlight,ReserveRoom,Confirmation,Cancellation\}}$$
$$ReserveFlight;(\overline{ConfirmFlight} + \overline{CancelFlight})$$

and

$$ReserveRoom;\overline{CancelRoom}$$
$$\preceq_{\{Reservation,ConfirmFlight,CancelFlight,ReserveFlight,ReserveRoom,Confirmation,Cancellation\}}$$
$$ReserveRoom;(\overline{ConfirmRoom} + \overline{CancelRoom})$$

We complete our examples of refinements with a travel agency contract that, differently from the contract in the Example 2.1, exploits only two local names $okFlight_*$ and $koFlight_*$ because the reply to the client is sent directly from the continuation of the subprocess interacting with the hotel reservation service. Also in this case, using the Theorem 3.3, it is possible to prove that

$$\begin{aligned}
&Reservation;\big(\overline{ReserveFlight};(ConfirmFlight;\overline{okFlight_*} + CancelFlight;\overline{koFlight_*}) \mid \\
&\quad \overline{ReserveRoom};(ConfirmRoom;(okFlight_*;\overline{Confirmation} + \\
&\qquad koFlight_*;\overline{Cancellation})) + \\
&\qquad (CancelRoom;(okFlight_*;\overline{Cancellation} + \\
&\qquad koFlight_*;\overline{Cancellation}))\big) \\
&\preceq_{\{Reservation,ConfirmFlight,CancelFlight,ConfirmRoom,CancelRoom\}} \\
&Reservation;\big(\overline{ReserveFlight};(ConfirmFlight;\overline{okFlight_*} + CancelFlight;\overline{koFlight_*}) \mid \\
&\quad \overline{ReserveRoom};(ConfirmRoom;\overline{okRoom_*} + CancelRoom;\overline{koRoom_*})\big) \mid \\
&\quad \big(okFlight_*;(okRoom_*;\overline{Confirmation} + koRoom_*;\overline{Cancellation}) + \\
&\quad koFlight_*;(okRoom_*;\overline{Cancellation} + koRoom_*;\overline{Cancellation})\big)
\end{aligned}$$

We conclude our discussion of this example observing that replacing the contracts in the correct composition of the Example 2.1 with the refinements discussed above, we obtain the following contract

composition which is guaranteed to be correct (by definition of subcontract pre-order)

$$
\begin{array}{l}
[\overline{Reservation}; (Confirmation + Cancellation + Retry; \overline{Reservation})] \parallel \\
[Reservation; (\overline{ReserveFlight}; (ConfirmFlight; \overline{okFlight_*} + CancelFlight; \overline{koFlight_*}) \mid \\
\qquad \overline{ReserveRoom}; (ConfirmRoom; (okFlight_*; \overline{Confirmation} + \\
\qquad\qquad koFlight_*; \overline{Cancellation})) + \\
\qquad\qquad (CancelRoom; (okFlight_*; \overline{Cancellation} + \\
\qquad\qquad\qquad koFlight_*; \overline{Cancellation}))] \parallel \\
[Reserve; \overline{ConfirmFlight}] \parallel \\
[Reserve; \overline{CancelRoom}]
\end{array}
$$

4. Conclusion and Future Work

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 significant technical differences between this paper and our previous papers [5, 4]. 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 3.2 and all its preliminary results), on the other hand, allow us to achieve a more general refinement that permits also to reduce external nondeterminism. Consider, for instance, the example reported at the beginning of Section 3, in particular the fact that $a + b \preceq_{\mathcal{N}-c} a + b + c$. 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] 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'. This result does not hold in this paper as proved by the counterexample reported after Proposition 3.5. A final remark, is concerned with the paper [4] 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 $\overline{a}$ 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], it is easy to prove that $(\overline{a};\overline{b}) + (\overline{b};\overline{a})$ is a strong subcontract of $\overline{a}|\overline{b}$, while the former is not (standard) compliant with the contract $a;b$ which is compliant with the latter.

We conclude commenting some future plans. We intend to investigate the connection between the calculi used in this paper and calculi for service choreography such as those presented in [7] and [9]. In particular, in [9] an end-point calculus similar to our contract calculus is considered where external choices must be guarded on input operations. Moreover, a subtyping relation is used in [9] to formalize similar aspects: the addition of input guarded branches in external choices is safe as well as the cancellation of output guarded branches in internal choices. Differently from [9] we consider a weaker *should-testing* semantics instead of the more restrictive (bi)simulation approach of [9]. This permits us, for instance, to abstract away from branching information that reveals not significant for contract composition.

References

[1] Robert Allen and David Garlan. Formalizing Architectural Connection. In *ICSE'94*, pages 71-80. IEEE Computer Society Press, 1994.

[2] Marco Autili, Paola Inverardi, Alfredo Navarra, and Massimo Tivoli. SYNTHESIS: a tool for automatically assembling correct and distributed component-based systems. In *ICSE'07*, pages 784–787. IEEE Computer Society Press, 2007.

[3] Mario Bravetti and Gianluigi Zavattaro. Contract based Multi-party Service Composition. In *FSEN'07*, volume 4767 of LNCS, pages 207–222. Springer, 2007.

[4] Mario Bravetti and Gianluigi Zavattaro. A Theory for Strong Service Compliance. In *Coordination'07*, volume 4467 of LNCS, pages 96–112. Springer, 2007.

[5] Mario Bravetti and Gianluigi Zavattaro. Towards a Unifying Theory for Choreography Conformance and Contract Compliance. In *SC'07*, volume 4829 of LNCS, pages 34–50. Springer, 2007.

[6] Antonio Brogi, Carlos Canal, and Ernesto Pimentel. Component adaptation through flexible subservicing. *Science of Computer Programming*, volume 63: 39–56. Elsevier, 2006.

[7] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration: A synergic approach for system design. In *ICSOC'05*, volume 3826 of LNCS, pages 228–240. Springer, 2005.

[8] Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and orchestration conformance for system design. In *Coordination'06*, volume 4038 of LNCS, pages 63–81. Springer, 2006.

[9] Marco Carbone, Kohei Honda, and Nabuko Yoshida. Structured Communication-Centred Programming for Web Services. In *ESOP'07*, volume 4421 of LNCS, pages 2–17. Springer, 2007.

[10] Marco Carbone, Kohei Honda, Nobuko Yoshida, Robin Milner, Gary Brown, and Steve Ross-Talbot. A Theoretical Basis of Communication-Centred Concurrent Programming. WCD-Working Note, 2006. Available at: http://www.dcs.qmul.ac.uk/∼carbonem/cdlpaper/workingnote.pdf

[11] Samuele Carpineti, Giuseppe Castagna, Cosimo Laneve, and Luca Padovani. A Formal Account of Contracts for Web Services. In *WS-FM'06*, volume 4184 of LNCS, pages 148-162. Springer, 2006.

[12] Samuele Carpineti and Cosimo Laneve A Basic Contract Language for Web Services. In *ESOP'06*, volume 3924 of LNCS, pages 197–213. Springer, 2006.

[13] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A Theory of Contracts for Web Services. In *POPL'08*, pages 261–272. ACM Press, 2008.

[14] Rocco De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, volume 24(2):211–237. Springer, 1887.

[15] Rocco De Nicola and Matthew Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, volume 34: 83–133. Elsevier, 1984.

[16] Cédric Fournet, C. A. R. Hoare, Sriram K. Rajamani, and Jakob Rehof. Stuck-Free Conformance. In *CAV'04*, volume 3114 of LNCS, pages 242–254. Springer, 2004.

[17] T. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[18] Naoki Kobayashi. Type Systems for Concurrent Processes: From Deadlock-Freedom to Livelock-Freedom, Time-Boundedness. In *IFIP TCS'00*, volume 1872 of LNCS, pages 365–389. Springer 2000.

[19] Cosimo Laneve and Luca Padovani. The must preorder revisited - An algebraic theory for web services contracts. In *Concur'07*, volume 4703 of LNCS, pages 212-225. Springer, 2007.

[20] Radu Mateescu, Pascal Poizat, and Gwen Salaün. Behavioral adaptation of component compositions based on process algebra encodings. In *ASE'07*, pages 385–388. ACM Press, 2007.

[21] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[22] OASIS. *WS-BPEL: Web Services Business Process Execution Language Version 2.0*. Technical report, OASIS, 2003.

[23] Arend Rensink and Walter Vogler. Fair testing. *Information and Computation*, volume 205: 125–198. Elsevier, 2007.

[24] W3C. *WS-CDL: Web Services Choreography Description Language*. Technical report, W3C, 2004.