

A Theory of Contracts for Strong Service Compliance

MARIO BRAVETTI¹ and GIANLUIGI ZAVATTARO¹

¹ *Department of Computer Science, Università di Bologna, {bravetti,zavattar}@cs.unibo.it*

Received 17 June 2008; Revised 16 February 2009

We investigate, in a process algebraic setting, a new notion of correctness for service compositions that we call *strong service compliance*: composed services are strong compliant if their composition is both deadlock and livelock free (this is the traditional notion of compliance) and whenever a message can be sent to invoke a service, this service is ensured to be ready to serve the invocation. We define also a new notion of refinement, called *strong subcontract pre-order*, suitable for strong compliance: given a composition of strong compliant services, we can replace any service with any other service in subcontract relation preserving the overall strong compliance. Finally, we present a characterization of the strong subcontract pre-order resorting to the theory of (should) testing pre-order.

1. Introduction

One of the main novelties emerged during the last years of research in the field of distributed computing is Service Oriented Computing (SOC). It is a novel paradigm 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 architecture*. 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 to include, in the exchanged messages, the so-called correlation information that permits to a service to associate a received message to the correct session of interaction (in fact, the same service could be contemporarily involved in different sessions at the same time).

Two main approaches for the composition of services are currently under investigation and development inside the SOC research community: service *orchestration* and service *choreography*. According to the first approach, the activities of the composed services are coordinated by a specific component, called the orchestrator, that is responsible for invoking the composed services and collect their responses. Several languages have

been already proposed for programming orchestrators such as XLANG (Thatte 2001), WSFL (Leymann 2001) and WS-BPEL (OASIS 2003).

Choreography languages are attracting a lot of attention within W3C, where the most credited choreography language WS-CDL (W3C 2004) is currently under development. Choreographies represent a “more democratic” alternative approach for service composition with respect to orchestrations. Indeed, orchestrations require the implementation of central points of coordination. On the contrary, choreography languages support a high level description of peer-to-peer interactions among services that directly communicate without the mediation of any orchestrator. More precisely, the aim of choreography languages is to support the high level description of systems that should be actually implemented as combination of autonomous, loosely coupled and heterogenous services.

One of the most challenging problems concerned with high-level specification languages for service composition is related with the discovery of available services that, once combined, are guaranteed to correctly implement the specified system. In order to solve this problem two aspects must be addressed: on the one hand, it is necessary to define appropriate interface languages that describe the externally observable behaviour of services; on the other hand, retrieval mechanisms must be developed such that, given a composition of services exposing certain interfaces, the overall behaviour of the system is guaranteed to behave according to the given high-level specification.

As far as interface languages are concerned, the attention of the research community is focusing on *contracts*, intended as the description of “the externally visible message-passing behaviour of the service” (Fournet *et al.* 2004). Several papers have already investigated contracts (Carpineti *et al.* 2006; Bravetti and Zavattaro 2007a; Laneve and Padovani 2007; Castagna *et al.* 2008), exploiting them in order to check the service compliance. Services are compliant when they can be correctly combined in such a way that all the service invocations of one service in the system are guaranteed to be eventually served by another service in the system.

For instance, the following three services S_1 , S_2 , and S_3 are compliant:

$$\begin{aligned} S_1 &: \text{invoke}(a@S_2); \text{receive}(b) \\ S_2 &: \text{receive}(a); \text{invoke}(c@S_3) \\ S_3 &: \text{receive}(c); \text{invoke}(b@S_1) \end{aligned}$$

We use $\text{invoke}(a@S)$ to denote the invocation of operation a on service S , while $\text{receive}(a)$ is the receipt of an invocation on operation a .

According to this notion of compliance, also the following is an example of compliant services:

$$\begin{aligned} S_1 &: \text{invoke}(a@S_2); \text{invoke}(b@S_3) \\ S_2 &: \text{receive}(a); \text{invoke}(c@S_3) \\ S_3 &: \text{receive}(c); \text{receive}(b) \end{aligned}$$

It is worth observing that the second invocation of S_1 (i.e. $\text{invoke}(b@S_3)$) cannot be served immediately, because it is necessary to wait for the interaction between S_2 and S_3 .

We can generalize this example to n services as follows:

$$\begin{aligned}
S_1 &: \text{invoke}(a_1@S_2); \text{invoke}(a_n@S_n) \\
S_2 &: \text{receive}(a_1); \text{invoke}(a_2@S_3) \\
S_3 &: \text{receive}(a_2); \text{invoke}(a_3@S_4) \\
&\dots \\
S_i &: \text{receive}(a_{i-1}); \text{invoke}(a_i@S_{i+1}) \\
&\dots \\
S_n &: \text{receive}(a_{n-1}); \text{receive}(a_n)
\end{aligned}$$

According to the above traditional notion of compliance also these services are compliant, even if the second invocation of S_1 must wait for $n - 1$ subsequent interactions among $n - 1$ different services. This arbitrary delay could be problematic if we consider protocols for service invocations delivery that exploit time-outs in order to avoid clients to wait indefinitely for the delivery of an invocation. In this case, the second invocation of S_1 will be almost surely timed-out. The occurrence of this time-out is clearly undesired while executing compliant services.

Notice that, in this reasoning, we are assuming service invocations to be “connection” based, i.e. requiring the invoked service to acknowledge data sent through the connection, (as it would happen, e.g., in a SOAP over HTTP implementation), and that we check the successful interaction of the services at the orchestration level only, where a connection by an invoke to an unavailable receive results in a (orchestration level) connection time-out failure, e.g. an exception. More precisely, we assume that network protocols are “perfect” at the connection management level, i.e. the lower levels of the stack of network protocols are able to manage possible network delays when performing a connection. This entails assuming that, if we invoke an operation of a service on which it is ready to receive, then the lower levels of the stack of protocols take care that no time-out failure is experienced (as in the first example above where services are undoubtedly compliant). In other words, we are focussing on the problem of capturing time-out failures that are due to delays at the level of the service behaviors (orchestration) only, where we assume that any unspecified amount of time can pass between the actual execution of an action and the subsequent one and unprefix receive actions are immediately available for the invoking service. As a consequence for services to be compliant we must have the logical guarantee that, when we perform an invoke on some operation, the invoked service is immediately ready to receive on that operation (for instance in the second example above we should add a communication from S_3 to S_1 that signals that S_3 is ready to receive on b). Note that, under the above assumptions, checking such immediate availability of receive operations is all we have to do to guarantee that no time-out failures occur in real orchestrations, hence we do not need to explicitly represent time in our models.

It is interesting to note that it is not necessary to consider an unbounded number of interacting services in order to model service compositions in which an output actions must wait for an unbounded number of interactions before being consumed by the expected

target service. Consider, for instance, the two following services:

$$\begin{aligned} S_1 &: \text{repeat}\{\text{invoke}(a@S_2); \text{receive}(b)\} \mid \text{invoke}(c@S_2) \\ S_2 &: \text{repeat}\{\text{receive}(a); \text{invoke}(b@S_1)\}; \text{receive}(c) \end{aligned}$$

where we assume that $\text{repeat}\{Op\}$ is a program construct used to execute the operation Op an arbitrary amount of times (corresponding to the Kleene star repetition operator $_*$ used in process algebras) and the vertical bar denotes the typical parallel composition of process algebras. In this example, the initially executable invocation $\text{invoke}(c@S_2)$ could wait for an unbounded amount of interactions executed by the operations inside the $\text{repeat}\{ \}$ constructs.

One could argue that the problem of time-outs can be solved associating to each service an input message queue in which the received messages are stored waiting for their actual delivery. Nevertheless, we note that also in this case the traditional notion of compliance could be not completely satisfactory. Consider, for example, the three following services:

$$\begin{aligned} S_1 &: \text{repeat}\{\text{invoke}(a@S_2)\}; \text{invoke}(b@S_3); \text{receive}(d) \\ S_2 &: \text{receive}(c); \text{repeat}\{\text{receive}(a)\}; \text{invoke}(d@S_1) \\ S_3 &: \text{receive}(b); \text{invoke}(c@S_2) \end{aligned}$$

In this example, we have that the input message queue of S_2 should have an unbounded capacity. This is clearly too demanding when, for instance, the services are executed within small portable devices with a limited storage capacity.

Paper contribution. As discussed above, the traditional notion of service compliance considered so far in the literature (see e.g. (Carpinetti *et al.* 2006; Bravetti and Zavattaro 2007a; Laneve and Padovani 2007; Castagna *et al.* 2008)) is not always satisfactory. In this paper, we address this lack of the previous formalizations of compliance, proposing the new *strong service compliance*. Intuitively, we require that, when a service is ready to execute an invocation on another service in the system, the target service must be ready to serve the invocation. According to this requirement, the services in the last four examples above are no longer compliant.

Besides proposing a formalization for strong service compliance, we present a theory for contracts which is consistent with strong service compliance. We follow a general novel approach for developing this theory. First of all, we proceed in a language independent manner representing the service behavior with a generic labeled transition system (with input, output, internal, and successful termination actions). In this way, our theory is applicable to any service description language assuming that it is equipped, or it can be equipped, with an operational semantics defined in terms of a labeled transition system. Moreover, we do not make any assumption neither on the syntax nor on the semantics for service description. This is not the case in the previous contract theories. For instance, in (Bravetti and Zavattaro 2007a) every output action is always syntactically preceded by an internal τ action while in (Carpinetti *et al.* 2006; Laneve and Padovani 2007; Castagna *et al.* 2008) an ad-hoc operational semantics for action prefixing is considered that hides internal choices.

Another interesting peculiarity of our theory is that we follow a general approach for

capturing the precise information to be extracted from the service behavior in order to be included in its service contract. This is achieved following 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. Once formalized the notion of strong service compliance, this refinement can be simply defined as follows: a service S' is a refinement of another service S if S' is compliant with all the services that are compliant with S .

One of the main results proved in this paper is that the achieved refinement is a pre-congruence, that is, given a set of compliant services, the refinement of one of these services is compliant not only with the other services in the system (this directly follows from the way we define the refinement), but also with all possible refinements of the other services. This result has not only a theoretical relevance, but we also foresee very important practical applications in the context of *service discovery* and *service update*.

As far as service discovery is concerned, we can consider 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 querying service registries collecting those services that either expose the expected contract, or one of its refinement. As the refinement is a pre-congruence, we can retrieve services independently one from the other (possibly in parallel) without breaking compliance. Independent retrieval is not supported by other theories for service contracts such as, for instance, the standard (i.e. not strong) subcontract relation proposed in (Laneve and Padovani 2007).

As far as *service update* is concerned, we can use strong compliance to ensure backward compatibility. Consider, e.g., a service that should be updated in order to provide new functionalities; if the new version is a refinement of the previous service, our theory ensures that the new service is a correct substitute for the previous one in any previously defined service compositions.

The last technical contribution of the paper is a characterization of our notion of refinement obtained resorting to the theory of testing (DH84), in particular to the should testing pre-order investigated in (Rensink and Vogler 2005). On the one hand, this characterization allows us to define an effective procedure to prove whether a service is a refinement of another one. On the other hand, we obtain a precise and formal comparison between our new refinement and the testing pre-order, which is the most similar theory available in the literature.

Structure of the paper. Section 2 reports the syntax and the operational semantics of the calculus that we use to formalize our theory. Section 3 reports the formalization of strong service compliance while Section 4 reports about our investigation of the refinement that we call strong subcontract relation. Section 5 contains conclusive remarks and a comparison with the related literature. Finally, Appendices A and B contain technical

machinery about: equivalence between model-based and process algebraic representation of contracts and the proof of the theorem characterizing our notion of refinement by resorting to the theory of testing.

2. Contracts and Service Compositions

In this section we introduce the syntax and the operational semantics of the calculi that we use in the following section to investigate formally the notion of strong compliance. The first calculus is a syntactical representation of generic labeled transition systems (with input, output, internal, and successful termination actions). Even if we call this first calculus the *contract* calculus, it is intended as a general model for the representation of the behaviors of services. For instance, we can represent with one contract the externally observable message passing behavior of the parallel composition of several services.

The second calculus (defined as an extension of the first one) allows services to be composed in order to form *systems* of interacting services. More precisely, a system is the parallel composition of contracts, each one with an associated location. The meaning of such location does not mandatorily coincide with the physical location where the services represented by the contract are running. More generally, contract locations are intended as logical roles that the services represented by the contract play inside a specified system. In other terms, we can see a system as a set of roles each one with an associated contract that describes the externally observable message passing behavior of the (possibly multiple) services that actually play such role.

For instance, we can specify a system with two logical locations/roles, a *client* and a *travel agency* offering a “plane and hotel reservation service”. At the contract level, it is sufficient to define the contract of the client and the contract of the travel agency. Given the contract of the travel agency, this could be physically implemented in different ways, either with a monolithic software running on the host of the travel agency, or in a distributed manner using several software components running on different physical host, e.g, an orchestrator running on the host of the travel agency, a plane reservation service running on some host of the Alitalia airplane company, and a reservation service running on some host of the Best Western hotel chain.

2.1. Contracts

Contracts are defined as labeled transition systems over located action names.

Definition 2.1. A finite connected labeled transition system (LTS) with termination transitions is a tuple $\mathcal{T} = (S, \mathcal{L}, \longrightarrow, s_h, s_0)$ where S is a finite set of states, \mathcal{L} is a set of labels, the transition relation \longrightarrow is a finite subset of $(S - \{s_h\}) \times (\mathcal{L} \cup \{\sqrt{\quad}\}) \times S$ such that $(s, \sqrt{\quad}, s') \in \longrightarrow$ implies $s' = s_h$, $s_h \in S$ represents a halt state, $s_0 \in S$ represents the initial state, and it holds that every state in S is reachable (according to \longrightarrow) from s_0 .

In a finite connected LTS with termination transitions we use $\sqrt{\quad}$ transitions (leading to the halt state s_h) to represent successful termination. On the contrary, if we get (via a

transition different from \surd) into a state with no outgoing transitions (like, e.g., s_h) then we represent an internal failure or a deadlock.

We assume a denumerable set of action names \mathcal{N} , ranged over by a, b, c, \dots and a denumerable set Loc of location names, ranged over by l, l', l_1, \dots . The set $\mathcal{N}_{loc} = \{a_l \mid a \in \mathcal{N}, l \in Loc\}$ is the set of located action names. We use $\tau \notin \mathcal{N}$ to denote an internal (unsynchronizable) computation. In contracts the possible transition labels are the typical internal τ action and the input/output actions a, \bar{a}_l , where the outputs (as we will see when composing contracts) are directed to a destination address denoted by a location $l \in Loc$.

Definition 2.2. A contract is a finite connected LTS with termination transitions, that is the tuple $(S, \mathcal{L}, \longrightarrow, s_h, s_0)$, where $\mathcal{L} = \{a, \bar{a}_l, \tau \mid a \in \mathcal{N}, l \in Loc\}$, i.e. labels are either a receive (input) on some operation $a \in \mathcal{N}$ or an invoke (output) directed to some operation $a \in \mathcal{N}$ at some location l .

In the following we introduce a process algebraic representation for contracts by using a basic process algebra (a simple extension of basic CCS (Milner 89) with successful termination) with prefixes over $\{a, \bar{a}_l, \tau \mid a \in \mathcal{N}, l \in Loc\}$ and we show that from the LTS denoting a contract we can derive a process algebraic term whose behavior is the same as that of the LTS. In the algebra syntax, we use $\mathbf{0}$ and $\mathbf{1}$ to denote unsuccessful and successful termination, respectively.

Definition 2.3. We consider a denumerable set of contract variables Var ranged over by X, Y, \dots . The syntax of contracts is defined by the following grammar

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

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

The structured operational semantics of contracts is defined in terms of a transition system labeled by $\mathcal{L} = \{a, \bar{a}_l, \tau, \mid a \in \mathcal{N}, l \in Loc\}$ obtained by the rules in Table 1 (plus symmetric rule for choice), where we take λ to range over $\mathcal{L} \cup \{\surd\}$. In particular the semantics of a contract $C \in \mathcal{P}_{con}$ gives rise to a finite connected LTS with termination transitions $(S, \mathcal{L}, \longrightarrow, \mathbf{0}, C)$ where S is the set of states reachable from C and \longrightarrow includes only transitions between states of S . Note that the fact that such a LTS is finite (i.e. finite-state and finitely branching) is a well-known fact for basic CCS (Milner 89) (and obviously the additional presence of successful termination does not change this fact).

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

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

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

2.2. Composing contracts

Definition 2.4. The syntax of systems (contract compositions) is defined by the following grammar

$$P ::= [C]_l \mid P \parallel P \mid P \setminus L$$

where $L \subseteq \mathcal{N}_{loc} \cup \{\bar{a}_l \mid a_l \in \mathcal{N}_{loc}\}$. A system P is well-formed if: (i) contract locations are unique, i.e. given distinct subterms of P $[C]_l$ and $[C]_{l'}$ we have that $l \neq l'$, and (ii) no output action with destination l is syntactically included inside a contract subterm occurring in P at the same location l , i.e. actions \bar{a}_l cannot occur inside a subterm $[C]_l$ of P . The set of all well-formed systems P is denoted by \mathcal{P} . In the following we will just consider well-formed systems and, for simplicity, we will call them just systems.

The syntax of compositions permits us to represent a service located at location l , and executing according to the contract C , simply as $[C]_l$. Services are composed using parallel composition \parallel and restriction \setminus . The restriction operator for compositions distinguishes between input and output actions, e.g., we write $[C]_l \setminus \{a_l, \bar{b}_{l'}\}$ to state that the service $[C]_l$ cannot perform inputs on a (e.g., because a is an output port of the service running at l) and cannot perform outputs on the port b of the service running at location l' (e.g., because b is an output port of that service).

Example 2.5. The first two service compositions discussed in the Introduction (those composed by three interacting services S_1 , S_2 , and S_3), are used as running examples in the paper. Their formalization in the calculus is as follows:

$$P_1 = [\bar{a}_{S_2}.b.\mathbf{1}]_{S_1} \parallel [a.\bar{c}_{S_3}.\mathbf{1}]_{S_2} \parallel [c.\bar{b}_{S_1}.\mathbf{1}]_{S_3}$$

$$P_2 = [\bar{a}_{S_2}.\bar{b}_{S_3}.\mathbf{1}]_{S_1} \parallel [a.\bar{c}_{S_3}.\mathbf{1}]_{S_2} \parallel [c.b.\mathbf{1}]_{S_3}$$

The operational semantics of systems is defined in terms of a transition system labeled by $\mathcal{L} = \{a_l, \bar{a}_l, \tau, \mid a \in \mathcal{N}, l \in Loc\}$ (with respect to contracts now also input labels are endowed with a location: the location of the receiver) obtained by the rules in Table 2 (plus symmetric rules), where we take λ to range over $\mathcal{L} \cup \{\surd\}$. It is defined in a standard way: the unique nonstandard operator is restriction that, as discussed above, distinguishes between input and output actions executed on the same name.

In the remainder of the paper we use the following notations: $P \xrightarrow{\lambda}$ (resp. $P \not\xrightarrow{\lambda}$) to mean that there exists P' such that $P \xrightarrow{\lambda} P'$ (resp. there exists no P' such that $P \xrightarrow{\lambda} P'$) and, given a sequence of labels $w = \lambda_1 \lambda_2 \cdots \lambda_{n-1} \lambda_n$ (possibly empty, i.e., $w = \varepsilon$), we use $P \xrightarrow{w} P'$ to denote the sequence of transitions $P \xrightarrow{\lambda_1} P_1 \xrightarrow{\lambda_2} \cdots \xrightarrow{\lambda_{n-1}} P_{n-1} \xrightarrow{\lambda_n} P'$ (in

$$\begin{array}{ccc}
\frac{C \xrightarrow{a} C'}{[C]_l \xrightarrow{a_l} [C']_l} & \frac{C \xrightarrow{\lambda} C' \quad \lambda = \bar{a}_l, \tau, \surd}{[C]_l \xrightarrow{\lambda} [C']_l} & \frac{P \xrightarrow{\lambda} P' \quad \lambda \neq \surd}{P \parallel Q \xrightarrow{\lambda} P' \parallel Q} \\
\frac{P \xrightarrow{a_l} P' \quad Q \xrightarrow{\bar{a}_l} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'} & \frac{P \xrightarrow{\surd} P' \quad Q \xrightarrow{\surd} Q'}{P \parallel Q \xrightarrow{\surd} P' \parallel Q'} & \frac{P \xrightarrow{\lambda} P' \quad \lambda \notin L}{P \setminus L \xrightarrow{\lambda} P' \setminus L}
\end{array}$$

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

case of $w = \varepsilon$ we have $P' = P$, i.e., $P \xrightarrow{\varepsilon} P$). We also adopt the usual notation A^* to denote (possibly empty) sequences over labels in a given set A . Finally, we use $P \xrightarrow{\lambda^*} P'$ to mean that P' is reachable from P via a (possibly empty) sequence of λ transitions.

Example 2.6. We discuss the operational semantics of the systems P_1 and P_2 defined in the Example 2.5 reporting some of their possible transitions. As far as P_1 is concerned, we consider the following computation:

$$\begin{array}{lcl}
P_1 & \xrightarrow{\tau} & [b.\mathbf{1}]_{S_1} \parallel [\bar{c}_{S_3}.\mathbf{1}]_{S_2} \parallel [c.\bar{b}_{S_1}.\mathbf{1}]_{S_3} & = P'_1 \\
& \xrightarrow{\tau} & [b.\mathbf{1}]_{S_1} \parallel [\mathbf{1}]_{S_2} \parallel [\bar{b}_{S_1}.\mathbf{1}]_{S_3} & = P''_1 \\
& \xrightarrow{\tau} & [\mathbf{1}]_{S_1} \parallel [\mathbf{1}]_{S_2} \parallel [\mathbf{1}]_{S_3} & \\
& \xrightarrow{\surd} & [\mathbf{0}]_{S_1} \parallel [\mathbf{0}]_{S_2} \parallel [\mathbf{0}]_{S_3} &
\end{array}$$

As far as P_2 is concerned, we simply observe that after one τ labeled transition, we achieve the following term P'_2 :

$$P_2 \xrightarrow{\tau} [\bar{b}_{S_3}.\mathbf{1}]_{S_1} \parallel [\bar{c}_{S_3}.\mathbf{1}]_{S_2} \parallel [c.b.\mathbf{1}]_{S_3} = P'_2$$

such that

$$P'_2 \xrightarrow{\bar{b}_{S_3}} [\mathbf{1}]_{S_1} \parallel [\bar{c}_{S_3}.\mathbf{1}]_{S_2} \parallel [c.b.\mathbf{1}]_{S_3} \quad \text{but} \quad P'_2 \not\xrightarrow{b_{S_3}}$$

This means that in P'_2 we have an output action (i.e. an invocation) on the name b of S_3 which has no corresponding input action (i.e. receive).

3. Strong Compliance

Intuitively, services/contracts are compliant when they can be combined in a correct composition, in which all service invocations are guaranteed to be served. According to this intuition, we are going to formalize the notion of *strong correct composition*. As discussed in the Introduction, in fact, typical notions of compliance requires that all invocations are guaranteed to be *eventually* served; our new stronger notion of compliance requires that all service invocations should be *immediately* served without waiting for the execution of other actions.

More precisely, in a strong correct composition it is guaranteed that all services eventually reach successful completion (the composition is both deadlock and livelock free) and everytime a process may invoke an operation on a service, the target service should be ready to serve the request. This second assumption is new and characterizes *strong compliance*.

As discussed in the Introduction, the rationale behind strong compliance is that standard protocols for service invocation usually raise exceptions in the case the target of a service invocation is not ready to serve it. In order to formalize situations in which exception cannot be raised, we define the auxiliary operator $\mathbf{nso}(P)$ that evaluates non-synchronizable outputs immediately executable by P , i.e. outputs that do not have a corresponding input, and the predicate $\mathbf{exceptionFree}(P)$ that indicates whether none of the above undesired exceptions can be raised in the system P .

Definition 3.1. (Exception freedom) We first define $\mathbf{nso}(P)$ inductively on the structure of P :

$$\begin{aligned} \mathbf{nso}([C]_l) &= \{\bar{a}_{l'} \mid C \xrightarrow{\bar{a}_{l'}} C'\} \\ \mathbf{nso}(P_1 \parallel P_2) &= (\mathbf{nso}(P_1) - \{\bar{a}_l \mid P_2 \xrightarrow{a_l} P'_2\}) \cup (\mathbf{nso}(P_2) - \{\bar{a}_l \mid P_1 \xrightarrow{a_l} P'_1\}) \\ \mathbf{nso}(P \setminus L) &= \begin{cases} \{\mathbf{exception}\} & \text{if } \mathbf{nso}(P) \cap L \neq \emptyset \\ \mathbf{nso}(P) & \text{otherwise} \end{cases} \end{aligned}$$

where $\mathbf{exception}$ is an auxiliary name denoting the existence of an output without a corresponding input. Finally, we define:

$$\mathbf{exceptionFree}(P) \text{ if and only if } \mathbf{nso}(P) = \emptyset$$

Definition 3.2. (Strong correct composition) A system P is a strong correct composition, denoted $P \Downarrow$, if for every P' such that $P \xrightarrow{\tau}^* P'$ both the following hold:

- $\mathbf{exceptionFree}(P')$ and
- there exists P'' such that $P' \xrightarrow{\tau}^* P'' \xrightarrow{\checkmark}$.

It is worth noting that strong compliance is a decidable property. In fact, contracts are finite labeled transition systems, thus also systems are finite as they are the parallel composition of a fixed set of contracts. The first item of the above definition is an instance of the reachability problem in a finite state system, thus it is decidable using standard algorithms. In order to check the second item, we can proceed as follows: we compute the set of successors of P (i.e. those states reachable from P via a sequence of τ -transitions), we compute the set of predecessors of the successfully terminated states (i.e. those states P' having at least one successor P'' such that $P'' \xrightarrow{\checkmark}$), and then we check that the first set is included in the second one.

Note also that, obviously, contracts that form correct contract compositions still form correct contract compositions if they are replaced by homomorphic ones.

Example 3.3. We continue the analysis of the systems P_1 and P_2 defined in the Example 2.5 checking whether they are strong correct compositions or not.

As far as P_1 is concerned, it is easy to see that the computation reported in the Example 2.6 is the unique possible sequence of τ labeled transitions (ending with a \checkmark labeled transition); moreover, for all intermediary states with output actions, namely P'_1 and P''_1 , we have that $\mathbf{nso}(P'_1) = \mathbf{nso}(P''_1) = \emptyset$ thus both $\mathbf{exceptionFree}(P'_1)$ and $\mathbf{exceptionFree}(P''_1)$ hold.

As far as P_2 is concerned, it is interesting to observe that its unique possible sequence of τ labeled transitions correctly ends with a \checkmark labeled transition, but the system P'_2

reached after the first transition is not exception free. Indeed, $\text{nso}(P'_2) = \{\bar{b}_{S_3}\}$. This implies that $P_2 \xrightarrow{\tau} P'_2$ with P'_2 such that $\text{exceptionFree}(P'_2)$ does not hold; thus, P_2 is not a strong correct composition.

Example 3.4. We now discuss examples in which we admit unbounded interactions among contracts. Consider the system $P = [C_1]_l \parallel [C_2]_{l'}$ with

$$C_1 = \text{rec}X.(\bar{a}_{l'}.X + \mathbf{1}) \quad C_2 = \text{rec}Y.(a.Y + \mathbf{1})$$

We have that the system P is correct (meaning a “strong correct composition”) because it is easy to see that all the configurations reachable with a sequence of τ actions are of the form $[C_1]_l \parallel [C_2]_{l'}$, and we have also that $[C_1]_l \parallel [C_2]_{l'} \xrightarrow{\surd} [\mathbf{0}]_l \parallel [\mathbf{0}]_{l'}$. On the contrary, if we assume

$$C_1 = \text{rec}X.(\bar{a}_{l'}.X + \mathbf{1}) \quad C_2 = \text{rec}Y.(a.Y + b.\mathbf{1})$$

we have that the system P is no longer correct because it is not possible for P to successfully complete (no \surd labeled transition can be executed by P or by one of its derivatives reachable executing only τ labeled transitions).

4. Contract Refinement

In this section we investigate a suitable notion of refinement for contracts compatible with strong correctness; intuitively, a contract C' is a *strong subcontract* of C if it is a “good” substitute of C , i.e. given a system P containing the service $[C]_l$, we can replace it with $[C']_l$ preserving the strong correctness of P .

4.1. Input-Output Strong Subcontract Relation

In general, see for instance (Carpinetti *et al.* 2006; Bravetti and Zavattaro 2007a), the subcontract relation depends on the alphabet (i.e. the possible actions) of the services present in C . For instance, we can consider $a + (b.C)$ subcontract of a assuming that the action \bar{b}_l is not in the alphabet of the other services (indeed, this implies that the new branch $b; C$ of the subcontract cannot interfere with the other services in the system breaking its strong correctness). We use this simple intuitive example of subcontract as a running example in this subsection.

We start defining a notion of subcontract parameterized on the input and output alphabets of the services in the potential contexts. Then we prove that, thanks to the new exception freedom assumption, we can abstract away from these alphabets.

We first formally define the input and output alphabets of systems. Given a set of located action names $I \subset \mathcal{N}_{loc}$, we denote: with $\bar{I} = \{\bar{a}_l \mid a_l \in I\}$ the set of output actions performable on those names and with $I_l = \{a \mid a_l \in I\}$ the set of action names with associated location l .

Definition 4.1. (Input and Output sets) Given the contract $C \in \mathcal{P}_{con}$, we define

$I(C)$ as the subset of \mathcal{N} of the potential input actions of C :

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

We define $O(C)$ as the subset of \mathcal{N}_{loc} of the potential output actions of C :

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

Given the system P , we define $I(P)$ as the subset of \mathcal{N}_{loc} of the potential input actions of P :

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

We define $O(P)$ as the subset of \mathcal{N}_{loc} of the potential output actions of P :

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

In the following we make the nonrestrictive assumption that the other services composed in parallel with the contract that we want to substitute with a subcontract are of the form $([C_1]_{l_1} \parallel \dots \parallel [C_n]_{l_n}) \setminus L$; this is not restrictive because it is always possible to use standard renaming techniques to avoid capture of names while extending the scope of restrictions. We denote with $\mathcal{P}_{conpres}$ the subset of systems of the form $([C_1]_{l_1} \parallel \dots \parallel [C_n]_{l_n}) \setminus L$.

Note that, given $P = ([C_1]_{l_1} \parallel \dots \parallel [C_n]_{l_n}) \setminus (I \cup \bar{O}) \in \mathcal{P}_{conpres}$, we have $I(P) = (\bigcup_{1 \leq i \leq n} I([C_i]_{l_i})) - I$ and $O(P) = (\bigcup_{1 \leq i \leq n} O([C_i]_{l_i})) - O$. In the following we let $\mathcal{P}_{conpres, I, O}$, with $I, O \subseteq \mathcal{N}_{loc}$, denote the subset of systems of $\mathcal{P}_{conpres}$ such that $I(P) \subseteq I$ and $O(P) \subseteq O$.

In the next definition we use the following notation: given a contract $C \in \mathcal{P}_{con}$, we use $oloc(C)$ to denote the subset of Loc of the locations target of the output actions occurring inside C .

Definition 4.2. (Input-Output strong subcontract relation) A contract C' is a subcontract of a contract C with respect to a set of input located names $I \subseteq \mathcal{N}_{loc}$ and output located names $O \subseteq \mathcal{N}_{loc}$, denoted $C' \preceq_{I, O} C$, if and only if for all $l \in Loc$ such that $l \notin oloc(C) \cup oloc(C')$ and $P \in \mathcal{P}_{conpres, I, O}$ such that $l \notin loc(P)$ we have

$$([C]_l \| P) \Downarrow \Rightarrow ([C']_l \| P) \Downarrow$$

As an example, we can formalize what already stated at the beginning of Section 4.1, that is, that $a + (b.C) \preceq_{\mathcal{N}_{loc}, \mathcal{N}_{loc} - \{b_l \mid l \in Loc\}} a$. This can be simply proved observing that if we substitute the service $[a]_l$ with $[a + (b.C)]_l$, in a context in which it is guaranteed that no invocations on the added b operation will be performed, the new service will interact with the context exactly as the initial service.

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 relation $\preceq_{I, O}$ (i.e. $\preceq_{I, O} \subseteq \preceq_{I', O'}$). This follows from the fact that extending the sets of input and output actions means considering a greater set of discriminating contexts.

Proposition 4.3. Let $C, C' \in \mathcal{P}_{con}$ be two contracts, $I, I' \subseteq \mathcal{N}_{loc}$ be two sets of input channel names such that $I' \subseteq I$ and $O, O' \subseteq \mathcal{N}_{loc}$ 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. Let us suppose $C' \preceq_{I,O} C$. Consider now $l \in Loc$, $l \notin oloc(C) \cup oloc(C')$, and $P \in \mathcal{P}_{conpres, I', O'}$, $l \notin loc(P)$, such that $([C]_l \| P) \Downarrow$. As $I' \subseteq I$ and $O' \subseteq O$, then also $P \in \mathcal{P}_{conpres, I, O}$. Thus, as we suppose $C' \preceq_{I,O} C$, $([C']_l \| P) \Downarrow$. This implies that also $C' \preceq_{I', O'} C$. \square

The following proposition states an intermediary result useful in subsequent proofs.

Proposition 4.4. Let $C, C' \in \mathcal{P}_{con}$ be contracts and $I, O \subseteq \mathcal{N}_{loc}$ be sets of located names and let $C' \preceq_{I,O} C$. For every $l \in Loc$, $l \notin oloc(C) \cup oloc(C')$, and $P \in \mathcal{P}_{conpres, I, O}$, $l \notin loc(P)$, such that $([C]_l \| P) \Downarrow$, we have

$$([C']_l \setminus (I([C']_l) - I([C]_l)) \| P) \Downarrow \quad \text{and} \quad ([C']_l \setminus (\overline{O(C')} - \overline{O(C)}) \| P) \Downarrow$$

Proof. We discuss the result concerned with restriction of outputs (the proof for the restriction of inputs is symmetric). Let $C' \preceq_{I,O} C$. Given any $P \in \mathcal{P}_{conpres, I, O}$ such that $([C]_l \| P) \Downarrow$, we will show that $([C']_l \setminus (\overline{O(C')} - \overline{O(C)}) \| P) \Downarrow$. As C does not perform actions on names outside its alphabet, we observe that $([C]_l \setminus P \setminus (O(C') - O(C))) \Downarrow$ because $([C]_l \setminus P) \Downarrow$. Since $C' \preceq_{I,O} C$, we derive $([C']_l \setminus P \setminus (O(C') - O(C))) \Downarrow$. As a consequence $([C']_l \setminus (\overline{O(C')} - \overline{O(C)}) \| P) \Downarrow$. We can conclude $([C']_l \setminus (\overline{O(C')} - \overline{O(C)}) \| P) \Downarrow$. \square

The following proposition states an important property which is a direct consequence of the assumption of exception freedom of correct compositions.

Proposition 4.5. Let $C, C' \in \mathcal{P}_{con}$ be contracts and $I, O \subseteq \mathcal{N}_{loc}$ be sets of located names and let $C' \preceq_{I,O} C$. For every $l \in Loc$, $l \notin oloc(C) \cup oloc(C')$, and $P \in \mathcal{P}_{conpres, I, O}$, $l \notin loc(P)$, such that $([C]_l \| P) \Downarrow$,

$$([C']_l \| P) \xrightarrow{\tau^*} ([C'_{der}]_l \| P_{der}) \quad \Rightarrow \quad \begin{cases} \forall a_{l'} \in O(C') - O(C). C'_{der} \xrightarrow{\bar{a}_{l'}} \\ \forall a \in I(C') - I(C). P_{der} \xrightarrow{\bar{a}} \end{cases}$$

Proof. We proceed by contradiction for both statements.

Concerning the first statement, suppose that there exist C'_{der} and P_{der} such that $([C']_l \| P) \xrightarrow{\tau^*} ([C'_{der}]_l \| P_{der})$ and $C'_{der} \xrightarrow{\bar{a}_{l'}}$ for some $a_{l'} \in O(C') - O(C)$. 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 $C'_{der2} \xrightarrow{\bar{a}_{l'}}$ for some $a_{l'} \in O(C') - O(C)$. This implies that the same path must be performable by $([C']_l \setminus (\overline{O(C')} - \overline{O(C)}) \| P)$, thus reaching the state $([C'_{der}]_l \setminus (\overline{O(C')} - \overline{O(C)}) \| P_{der})$. However, since in the state C'_{der} of contract C' we have $C'_{der} \xrightarrow{\bar{a}_{l'}}$ for some $a_{l'} \in O(C') - O(C)$ and the execution of $\bar{a}_{l'}$ is disallowed by restriction, we will have $\text{ns}([C'_{der}]_l \setminus (\overline{O(C')} - \overline{O(C)})) = \{\text{exception}\}$, thus $([C']_l \setminus (\overline{O(C')} - \overline{O(C)}) \| P) \not\Downarrow$ contradicting Proposition 4.4.

Concerning the second statement, suppose that there exist C'_{der} and P_{der} such that

$([C']_l \parallel P) \xrightarrow{\tau}^* ([C'_{der}]_l \parallel P_{der})$ and $P_{der} \xrightarrow{\bar{a}_l}$ for some $a \in I(C') - I(C)$. We further suppose (without loss of generality) that such a path is minimal, i.e. no intermediate state $(C'_{der2} \parallel P_{der2})$ is traversed, such that $P_{der2} \xrightarrow{\bar{a}_l}$ for some $a \in I(C') - I(C)$. This implies that the same path must be performable by $([C']_l \parallel P \parallel (I([C']_l) - I([C]_l)))$, thus reaching the state $([C'_{der}]_l \parallel P_{der} \parallel (I([C']_l) - I([C]_l)))$. However, since in the state P_{der} of system P we have $P_{der} \xrightarrow{\bar{a}_l}$ for some $a \in I(C') - I(C)$ and the execution of \bar{a}_l is disallowed by restriction, we will have $\text{nso}(P_{der} \parallel (I([C']_l) - I([C]_l))) = \{\text{exception}\}$, thus $([C']_l \parallel P \parallel (I([C']_l) - I([C]_l))) \not\Downarrow$. This implies $([C']_l \parallel (I([C']_l) - I([C]_l))) \not\Downarrow$ contradicting Proposition 4.4. \square

We are finally ready to prove the main results of this subsection. We separate these results in two independent propositions; the first one states that the set of potential inputs of the other contracts in the system is an information that does not influence the strong subcontract relation, the second one states the same about outputs.

Proposition 4.6. Let $C \in \mathcal{P}_{con}$ be a contract, $O \subseteq \mathcal{N}_{loc}$ be a set of located output names and $I, I' \subseteq \mathcal{N}_{loc}$ be two sets of located input names such that $O(C) \subseteq I, I'$. We have that for every contract $C' \in \mathcal{P}_{con}$,

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

Proof. Let us suppose $C' \preceq_{I',O} C$ (the other direction is symmetric). Given any $l \in Loc$, $l \notin oloc(C) \cup oloc(C')$, and $P \in \mathcal{P}_{conpres,I,O}$, $l \notin loc(P)$, such that $([C]_l \parallel P) \Downarrow$, we will show that $([C']_l \parallel P) \Downarrow$. We first observe that $([C]_l \parallel P \parallel (I - O(C))) \Downarrow$. Since $C' \preceq_{I',O} C$ and $O(C) \subseteq I'$, we derive $([C']_l \parallel P \parallel (I - O(C))) \Downarrow$. Due to Proposition 4.5 we have that $([C']_l \parallel P \parallel (I - O(C)))$ 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 $([C']_l \parallel P) \Downarrow$. \square

Proposition 4.7. Let $C \in \mathcal{P}_{con}$ be a contract, $O, O' \subseteq \mathcal{N}_{loc}$ be two sets of located output names such that for every $l \in Loc$ we have $I(C) \subseteq O_l, O'_l$, and $I \subseteq \mathcal{N}_{loc}$ be a set of located input names. We have that for every contract $C' \in \mathcal{P}_{con}$,

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

Proof. Let us suppose $C' \preceq_{I,O'} C$ (the other direction is symmetric). Given any $l \in Loc$, $l \notin oloc(C) \cup oloc(C')$, and $P \in \mathcal{P}_{conpres,I,O}$, $l \notin loc(P)$, such that $([C]_l \parallel P) \Downarrow$, we show that $([C']_l \parallel P) \Downarrow$. We observe that $([C]_l \parallel P \parallel (O - I([C]_l))) \Downarrow$. Since $C' \preceq_{I,O'} C$ and $I([C]_l) \subseteq O'$, we derive $([C']_l \parallel P \parallel (O - I([C]_l))) \Downarrow$. As a consequence $([C']_l \parallel I(C') - I(C) \parallel P \parallel (O - I([C]_l))) \Downarrow$ and $([C']_l \parallel I(C') - I(C) \parallel P) \Downarrow$. Due to Proposition 4.5 we have that $([C']_l \parallel I(C') - I(C) \parallel P)$ can never reach by τ transitions a state where outputs in $I([C']_l) - I([C]_l)$ are executable by some derivative of P , so we conclude $([C']_l \parallel P) \Downarrow$. \square

As a consequence of the last proposition, we have that the already proved relation

$$a + (b.C) \preceq_{\mathcal{N}_{loc}, \mathcal{N}_{loc} - \{b_l \mid l \in Loc\}} a$$

implies the more general relation

$$a + (b.C) \preceq_{\mathcal{N}_{loc}, \mathcal{N}_{loc}} a$$

Intuitively, this holds because given a correct system with a service $[a]_l$, it is guaranteed that no other service will be able to execute the output action \bar{b}_l (even if b_l is in its output alphabet), otherwise the overall system cannot be correct because this output action is surely non-synchronizable.

In general, these last two propositions permit us to forget about the restrictions on the input/output alphabets of the services in the context in which we apply the substitution of one contract with one of its subcontracts, considering always $\preceq_{\mathcal{N}_{loc}, \mathcal{N}_{loc}}$. We denote this relation simply with \preceq and we call it the *strong subcontract relation*. We define \preceq assuming a limited set of possible contexts and then we prove that this limitation is not relevant. The new set of contexts does not contain restrictions, i.e., we consider $[C_1]_{l_1} \parallel \dots \parallel [C_n]_{l_n}$ instead of $([C_1]_{l_1} \parallel \dots \parallel [C_n]_{l_n}) \setminus L$. We call \mathcal{P}_{compar} the subset of systems of the form $[C_1]_{l_1} \parallel \dots \parallel [C_n]_{l_n}$.

Definition 4.8. (Strong subcontract relation) A contract C' is a strong subcontract of a contract C denoted $C' \preceq C$, if and only if for all $l \in Loc$ such that $l \notin oloc(C) \cup oloc(C')$ and $P \in \mathcal{P}_{compar}$ such that $l \notin loc(P)$ we have

$$([C]_l \parallel P) \Downarrow \Rightarrow ([C']_l \parallel P) \Downarrow$$

We now prove that, even if we have considered in the definition of the strong subcontract relation only contexts without restrictions, we capture the same notion of subcontract previously defined considering all possible contexts.

Proposition 4.9. Let $C, C' \in \mathcal{P}_{com}$ be two contracts:

$$C \preceq C' \quad \text{if and only if} \quad C' \preceq_{\mathcal{N}_{loc}, \mathcal{N}_{loc}} C$$

Proof. The if part is simple as \preceq is defined as $\preceq_{\mathcal{N}_{loc}, \mathcal{N}_{loc}}$ assuming a subset of possible contexts P .

We now prove the only-if part. Suppose $[C]_l \parallel (P \setminus L)$ with $P = ([C_1]_{l_1} \parallel \dots \parallel [C_n]_{l_n})$; let $I, O \subset \mathcal{N}_{loc}$ be such that $I = \{a_{l_i} \in L \mid 1 \leq i \leq n\}$ and $O = \{a_l \mid \bar{a}_l \in L\}$ (in O only outputs on the location l are considered). We have that $([C]_l \parallel (P \setminus L)) \Downarrow \iff ([C]_l \parallel (P \setminus I \cup \bar{O})) \Downarrow \iff ([C]_l \parallel (P \setminus \{\tau.\mathbf{0}/\alpha \mid \alpha \in \bar{O}\} \setminus I)) \Downarrow \iff ([C]_l \parallel P') \Downarrow$, where P' is obtained from $P'' = P \setminus \{\tau.\mathbf{0}/\alpha \mid \alpha \in \bar{O}\}$ as follows. We call $M \in \mathcal{N}$ the (finite) set of action names occurring in C and C' . We consider an arbitrary injective function rel that maps each action name a in M into a fresh name $rel(a)$. For each $a_{l'} \in I$, we do the following: (i) we replace each syntactical occurrence of a inside the unique subterm $[C'']_{l'}$ of P'' with $rel(a)$, and (ii) we replace each syntactical occurrence of $\bar{a}_{l'}$ inside P'' with $rel(a)_{l'}$. Since the same chain of “ \iff ” holds for C' (using the same relabeling function “ rel ”), we have that the result is a direct consequence of the definition of strong subcontract relation applied to P' . \square

If we consider the running example of this subsection, we already observed that

$$a + (b.C) \preceq_{\mathcal{N}_{loc}, \mathcal{N}_{loc}} a$$

thus, in light of this last proposition, we have also that $a + (b.C) \preceq a$.

4.2. Independent Refinement

In this subsection we prove that the strong subcontract relation \preceq , which has been defined assuming that the other services in the context are kept unchanged while applying the refinement, is suitable also for a more general refinement that is applied independently on all services contemporarily. This is an important property for a refinement notion suitable for service oriented computing; indeed, services are loosely coupled in the sense that they can be updated/modified independently one from the other ones. Independent refinements can be defined as follows.

Definition 4.10. (Independent strong subcontract pre-order) A pre-order \leq over \mathcal{P}_{con} is an independent strong subcontract pre-order if, for any $n \geq 1$, contracts $C_1, \dots, C_n \in \mathcal{P}_{con}$ and $C'_1, \dots, C'_n \in \mathcal{P}_{con}$ such that $\forall i. C'_i \leq C_i$, and distinguished location names $l_1, \dots, l_n \in Loc$ such that $\forall i. l_i \notin oloc(C_i) \cup oloc(C'_i)$, we have

$$([C_1]_{l_1} \parallel \dots \parallel [C_n]_{l_n}) \Downarrow \Rightarrow ([C'_1]_{l_1} \parallel \dots \parallel [C'_n]_{l_n}) \Downarrow$$

We will show that the maximal independent strong subcontract pre-order corresponds to the relation \preceq defined in the previous subsection. This is achieved defining a more general class of pre-orders called *singular strong subcontract pre-orders*, observing that \preceq is the maximal singular strong subcontract pre-order, and finally showing that all independent strong subcontract pre-orders are also singular strong subcontract pre-orders, and vice versa.

Intuitively a pre-order \leq over \mathcal{P}_{con} is a singular strong subcontract pre-order whenever the strong correctness of systems is preserved by refining just one of the contracts. More precisely, for any $n \geq 1$, contracts $C_1, \dots, C_n \in \mathcal{P}_{con}$, $1 \leq i \leq n, C'_i \in \mathcal{P}_{con}$ such that $C'_i \leq C_i$, and distinguished location names $l_1, \dots, l_n \in Loc$ such that $\forall k \neq i. l_k \notin oloc(C_k)$ and $l_i \notin oloc(C_i) \cup oloc(C'_i)$, we require

$$([C_1]_{l_1} \parallel \dots \parallel [C_i]_{l_i} \parallel \dots \parallel [C_n]_{l_n}) \Downarrow \Rightarrow ([C_1]_{l_1} \parallel \dots \parallel [C'_i]_{l_i} \parallel \dots \parallel [C_n]_{l_n}) \Downarrow$$

By exploiting commutativity and associativity of parallel composition we can group the contracts which are not being refined and get the following cleaner definition. We recall that \mathcal{P}_{conpar} denotes the subset of systems of the form $[C_1]_{l_1} \parallel \dots \parallel [C_n]_{l_n}$.

Definition 4.11. (Singular strong subcontract pre-order) A pre-order \leq over \mathcal{P}_{con} is a singular strong subcontract pre-order if, for any $C, C' \in \mathcal{P}_{con}$ such that $C' \leq C$, $l \in Loc$ such that $l \notin oloc(C) \cup oloc(C')$, $P \in \mathcal{P}_{conpar}$ such that $l \notin loc(P)$ we have

$$([C]_l \parallel P) \Downarrow \Rightarrow ([C']_l \parallel P) \Downarrow$$

It is easy to see that the strong subcontract relation \preceq is the maximal singular strong subcontract pre-order as it relates all pairs of contracts that satisfy the property stated in the Definition 4.11.

In order to prove the existence of the maximal independent strong subcontract pre-order, we will prove that every pre-order that is an independent strong subcontract is also a singular strong subcontract (Theorem 4.12), and vice-versa (Theorem 4.13).

Theorem 4.12. If a pre-order \leq is an independent strong subcontract pre-order then it is also a singular strong subcontract pre-order.

Proof. Suppose that \leq is an independent strong subcontract pre-order. Consider $n \geq 1$, $C, C' \in \mathcal{P}_{con}$, $l \in Loc$ and $P \in \mathcal{P}_{compar}$ such that $l \notin loc(P)$. From $([C]_l \| P) \Downarrow$ and $C' \leq C$, we can derive $([C']_l \| P) \Downarrow$ by just taking in the definition of independent strong subcontract pre-order, $C_1 = C$, $C'_1 = C'$, $C_2 \dots C_n$ to be such that $P = (C_2 \| \dots \| C_n)$ and finally C'_i to be C_i for every $i \geq 2$ (since \leq is a pre-order we have $C \leq C$ for every C). \square

Theorem 4.13. If a pre-order \leq is a singular strong subcontract pre-order then it is also an independent strong subcontract pre-order.

Proof. Consider $n \geq 1$, contracts $C_1, \dots, C_n \in \mathcal{P}_{con}$ and $C'_1, \dots, C'_n \in \mathcal{P}_{con}$ such that $\forall i. C'_i \leq C_i$, and distinguished location names $l_1, \dots, l_n \in Loc$ such that $\forall i. l_i \notin oloc(C_i) \cup oloc(C'_i)$. For any i we let $P_i = [C_i]_{l_i}$ and $P'_i = [C'_i]_{l_i}$. If $(P_1 \| \dots \| P_n) \Downarrow$ we can derive $(P'_1 \| \dots \| P'_n) \Downarrow$ in n steps: at the i -th step we replace P_i with P'_i without altering the correctness of the system. \square

We can, therefore, conclude that “ \preceq ”, which is the maximal strong singular subcontract pre-order, is also the maximal independent strong subcontract pre-order.

4.3. Resorting to Should Testing

In order to check whether two contracts are in strong subcontract relation the Definition 4.8 is not usable in practice as it contains universal quantifications on every possible location l and context P . We now present an actual procedure for proving that two contracts are in strong subcontract relation achieved resorting to the theory of *should-testing* (Rensink and Vogler 2005).

We introduce a restriction operator on contracts as an abuse of notation: “ $C \setminus M$ ” stands for “ $C \{ \mathbf{0} / \alpha.C' \}$ for each $\alpha \in M$ ” with $M \subseteq \mathcal{N}$ (i.e. all terms prefixed by actions in M are replaced by $\mathbf{0}$). This allows us, e.g., to achieve a transition system isomorphic to that of $[C]_l \setminus I$, with $I = \{a_l \mid a \in M\}$ for some $M \subseteq \mathcal{N}$, simply by considering $[C \setminus M]_l$.

First, we need a preliminary result that is a direct consequence of the fact that

$$C' \preceq_{\mathcal{N}_{loc} \cup_{l \in Loc} I([C]_l)} C \text{ if and only if } C' \preceq C$$

due to Proposition 4.7.

Lemma 4.14. Let $C, C' \in \mathcal{P}_{con}$ be contracts. We have

$$C' \setminus (I(C') - I(C)) \preceq C \quad \Rightarrow \quad C' \preceq C$$

Proof. We will show that the hypothesis yields $C' \preceq_{\mathcal{N}_{loc} \cup_{l \in Loc} I([C]_l)} C$. From this we can derive the result by using Proposition 4.7. Given any $l \in Loc$, $l \notin oloc(C) \cup oloc(C')$,

and $P \in \mathcal{P}_{conpres, \mathcal{N}_{loc}, \cup_{l \in Loc} I([C]_l)}$, $l \notin loc(P)$, such that $([C]_l \| P) \Downarrow$, we will show that $([C']_l \| P) \Downarrow$. We have $([C' \setminus I(C') - I(C)]_l \| P) \Downarrow \iff ([C' \setminus I(C') - I(C)]_l \| P \setminus \overline{I([C']_l) - I([C]_l)}) \Downarrow$
 $\iff ([C']_l \| P \setminus \overline{I([C']_l) - I([C]_l)}) \Downarrow \iff ([C']_l \| P) \Downarrow$. \square

Note that the opposite implication trivially holds (by taking $O = \mathcal{N}_{loc}$ and $I = \mathcal{N}_{loc}$ in Proposition 4.4).

In the following we denote with \preceq_{test} the *should-testing* pre-order defined in (Rensink and Vogler 2005). Intuitively, two processes are in should-testing pre-order if the former satisfies all the *tests* satisfied by the latter. A test is a process capable to communicate success on a specific actions. In (Rensink and Vogler 2005) this action is \surd ; as we have already used \surd for contracts, we consider \surd' . According to the definition in (Rensink and Vogler 2005), a test t is satisfied by a process Q (written $Q \mathbf{shd} t$) if the following holds:

$$\forall w \in \mathcal{A}_\tau, Q'. \quad Q \parallel_{\mathcal{A}} t \xrightarrow{w} Q' \quad \Rightarrow \quad \exists v \in \mathcal{A}_\tau, Q'' : Q' \xrightarrow{v} Q'' \xrightarrow{\surd'}$$

where \mathcal{A}_τ is the alphabet \mathcal{A} of all possible actions in the considered process algebra plus τ and $\parallel_{\mathcal{A}}$ is the CSP parallel operator: in $R \parallel_{\mathcal{A}} R'$ 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 .

Should-testing has been defined as an enhancement of the classical must-testing (DH84) to deal also with fair computations. For instance, consider the process $Q = recX.(a.X + b)$ and the test $t = recX.(a.X + b.\surd')$ (for simplicity we have defined Q and t using our syntax instead of the slightly different syntax of prefix and recursion used in (Rensink and Vogler 2005) as we will see in the following). We have that Q satisfies the test t only under the should-testing approach (while this is not the case for must-testing) because the execution of \surd' is guaranteed only under the assumption of fairness.

In the following, we consider the set of actions \mathcal{A} used by terms as being $\mathcal{A} = \mathcal{N} \cup \mathcal{N}_{loc} \cup \{\bar{a}_l \mid a_l \in \mathcal{N}_{loc}\} \cup \{\surd\}$ (i.e. we consider located input and output actions, unlocated input actions and \surd : the latter is included in the set of actions of terms being tested as any other action). As stated above, we denote here with \surd' the special action for the success of the test (denoted by \surd in (Rensink and Vogler 2005)).

In order to resort to the theory defined in (Rensink and Vogler 2005), we first define a transformation on the finite labeled transition system (LTS) of a contract C .

Here we use quadruples $(S, Lab, \longrightarrow, s_{init})$ to represent generic LTSs, 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.

Hence, we can see the semantics $\llbracket C \rrbracket$ of a contract C as a generic LTS $\llbracket C \rrbracket = (S, \mathcal{A}_\tau, \longrightarrow, C)$, where S is the set of states reachable from C according to the transition relation defined by the operational rules for contracts in Table 1 and \longrightarrow is the subset of such a transition relation obtained by just considering transitions between states in S .

The transformation of $\llbracket C \rrbracket = (S, \mathcal{A}_\tau, \longrightarrow, C)$ is performed in two steps:

- 1 First, for every state s of the LTS we do the following: called $I(s)$ the set of labels of outgoing input transitions from the state s , for every label in $I(C) - I(s)$ we add

to the state s an outgoing input transition with that label that leads to a *new* state with no outgoing output transitions.

- 2 Then, for every output transition \bar{a}_l , we replace the output transition with a pair of transitions connected by an additional state (the first one has the same source as the previous output transition and the second one has the same destination as the previous output transition): a τ transition followed by an \bar{a}_l output transition.

Formally, we define the LTS $(S', \mathcal{A}_\tau, \longrightarrow', C)$ obtained by the transformation as follows:

$$\begin{aligned} \text{--- } S' &= S \cup \{new\} \cup \{(s, \bar{a}_l, s') \mid s \xrightarrow{\bar{a}_l} s'\} \\ \text{--- } \longrightarrow' &= \{(s, \lambda, s') \mid s \xrightarrow{\lambda} s' \wedge \lambda \in \mathcal{A}_\tau \wedge \nexists a, l : \lambda = \bar{a}_l\} \\ &\quad \cup \{(s, a, new) \mid a \in I(C) \wedge s \xrightarrow{a} \cdot\} \\ &\quad \cup \{(s, \tau, (s, \bar{a}_l, s')) \mid s \xrightarrow{\bar{a}_l} s'\} \cup \{((s, \bar{a}_l, s'), \bar{a}_l, s') \mid s \xrightarrow{\bar{a}_l} s'\} \end{aligned}$$

Here, we describe the effect of the two above transformations applied to the contracts checked in the testing scenario that we are going to formalize. The first transformation on the labeled transition system is used to capture those output actions performed by the tester for which there is no corresponding input action in the tested process; the new added input can synchronize with these actions and lead to the state *new*. This disallows the possibility for the tester to complete its execution, thus to succeed. The second transformation, on the other hand, is used to check whether all output actions performable by the tested contract can be received by the tester: in fact, the τ transition added before the output action makes it possible to enter in a state where only the output action is executable. If the tester does not consume this output the contract sticks and the test cannot succeed.

Now we derive a normal form for contracts of our calculus that corresponds to terms of the language in (Rensink and Vogler 2005). The normal form for a contract C (denoted with $\mathcal{NF}(C)$) is defined as follows, by using the operators “ $rec_X\theta$ ” and “ $\lambda; Q$ ” (defined in (Rensink and Vogler 2005)) that represent the value of X in the solution of the minimum fixpoint of the finite set of equations θ and prefixing of term Q by action λ , respectively,

$$\mathcal{NF}(C) = rec_{X_C}\theta$$

with θ being the set of S' -indexed equations

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

where we use es, es' (standing for “extended state”) to represent states of S' and we assume empty sums to be equal to $\mathbf{0}$, i.e. if there are no outgoing transitions from X_{es} , we have $X_{es} = \mathbf{0}$.

According to the definitions in (Rensink and Vogler 2005), the semantics $\llbracket \mathcal{NF}(C) \rrbracket$ of the normal form $\mathcal{NF}(C) = rec_{X_C}\theta$ is, as expected, the labeled transition system $\llbracket \mathcal{NF}(C) \rrbracket = (S'', \mathcal{A}_\tau, \longrightarrow'', \mathcal{NF}(C))$, where:

$$\begin{aligned} \text{--- } S'' &= \{\mathcal{NF}(es) = rec_{X_{es}}\theta \mid es \in S'\} \\ \text{--- } \longrightarrow'' &= \{(\mathcal{NF}(es), \lambda, \mathcal{NF}(es')) \mid es \xrightarrow{\lambda} es'\} \end{aligned}$$

In conclusion, given a contract C , we define $map(\llbracket C \rrbracket)$ to be the LTS $\llbracket \mathcal{NF}(C) \rrbracket$ defined above.

We are now in a position to define the sound characterization of the strong subcontract relation in terms of testing.

Theorem 4.15. Let $C, C' \in \mathcal{P}_{con}$ be two contracts. We have

$$\mathcal{NF}(C' \setminus I(C') - I(C)) \preceq_{test} \mathcal{NF}(C) \quad \Rightarrow \quad C' \preceq C$$

Proof. See Appendix B. □

We observe that this theorem introduces an actual procedure to prove whether two contracts are in subcontract relation. In fact, given two contracts C and C' we can prove that $C' \preceq C$ as follows:

- 1 construct the two terms $\mathcal{NF}(C' \setminus I(C') - I(C))$ and $\mathcal{NF}(C)$;
- 2 apply the algorithm presented in (Rensink and Vogler 2005) to check whether the former is a should-testing refinement of the latter.

We first note that the transformation $\mathcal{NF}(_)$ is not applied directly to C' but to $C' \setminus I(C') - I(C)$, i.e. to C' restricted on all those inputs that are not part of the input alphabet of C . This restriction permits us to add in C' new branches guarded by inputs on new names.

Example 4.16. Consider the simple contract $C = a$ and the contract $C' = a + b.C''$, for some contract C'' . If we restrict the latter on the names $I(C') - I(C) = \{b\}$ we obtain $(a + b.C'') \setminus \{b\}$. It is easy to see that the labeled transition system of this latter term is isomorphic to the one of the contract a . For this reason, we can conclude that

$$a + b.C'' \preceq a$$

As discussed above the function $\mathcal{NF}(_)$ performs two kinds of transformations on contracts. The first one consists of the addition to every state s of outgoing input transitions, on channels on which s has no already active inputs, leading to a deadlocked state. The second one is the addition of τ -labeled transitions before every output actions.

A consequence of the first transformation is that a subcontract, besides adding new branches guarded by inputs on new names (as already discussed), can add also new branches guarded by inputs on names that are already part of the alphabet. These additions can be done only in states in which the contract under refinement did not admit inputs on that particular name. A consequence of the second transformation is the possibility to remove, in a state performing a choice among outputs, branches guarded by some of these outputs.

Example 4.17. Consider the simple contract $C = b.a$ and the contract $C' = b.(a + b.0)$. If we perform the first kind of transformation discussed above on C and C' it is easy to see that we obtain isomorphic labeled transition systems. For this reason, we can conclude that

$$b.(a + b.0) \preceq b.a$$

Note that we can also conclude that

$$b.a \preceq b.(a + b.\mathbf{0})$$

Example 4.18. Consider the contract $C = \bar{a}_l + \bar{b}_l$ and the contract $C' = \bar{a}_l$. If we perform the second kind of transformation discussed above on C we obtain $\tau.\bar{a}_l + \tau.\bar{b}_l$, while if we perform the same transformation to C' we obtain $\tau.\bar{a}_l$. It is easy well known that the testing pre-order (thus also the should testing pre-order) allows a refined process to exhibit less internal nondeterminism. In general, we have that $\tau.P + \tau.Q$ can be refined by $\tau.P$. For this reason, we can conclude that

$$\bar{a}_l \preceq \bar{a}_l + \bar{b}_l$$

The last step of the algorithm described above consists of checking whether the terms obtained after transformation are in relationship according to should testing pre-order. In the following we show characteristic examples of terms that are related because we resort to such a pre-order.

Example 4.19. Consider the contract $C = a.C_1 + a.C_2$ for some contract C_1 and C_2 , and the contract $C' = a.C_1$. As discussed above, the should testing pre-order allows a refined process to exhibit less internal nondeterminism. In general, we have that $a.P + a.Q$ can be refined by $a.P$. For this reason, we can conclude that

$$a.C_1 \preceq a.C_1 + a.C_2$$

Example 4.20. The main difference between should testing pre-order and standard testing is in the fair treatment of divergence. For instance, the process $a.recX.(\tau.X + b)$ is equivalent to $a.b$ according to should-testing, while this is not the case in standard testing. For this reason, we can conclude that

$$a.recX.(\tau.X + b) \preceq a.b \quad \text{and} \quad a.b \preceq a.recX.(\tau.X + b)$$

The sound characterization of the strong subcontract relation in terms of should testing allows us to compare it with existing well-known relation, as, e.g., simulation (i.e. half-bisimulation) and (should) testing, and to state that (up to the presented transformation) it is coarser with respect to them. This, e.g., tells us that developing from scratch a notion of contract refinement that is bisimulation based would be indeed requiring too much.

Example 4.21. The classical example of difference between bisimulation and testing also applies to our context (in the case of output actions), that is, we have that

$$\bar{a}_l.(\bar{b}_l.\bar{c}_l + \bar{b}_l.\bar{d}_l) \preceq \bar{a}_l.\bar{b}_l.\bar{c}_l + \bar{a}_l.\bar{b}_l.\bar{d}_l \quad \text{and} \quad \bar{a}_l.\bar{b}_l.\bar{c}_l + \bar{a}_l.\bar{b}_l.\bar{d}_l \preceq \bar{a}_l.(\bar{b}_l.\bar{c}_l + \bar{b}_l.\bar{d}_l)$$

5. Conclusion and Related Work

We have considered a new notion of service compliance, modeled using process calculi, in which we assume that whenever a message is sent to a service in order to invoke a particular operation, the service is ready to serve it. We called this new notion *strong compliance*, and we have developed around it an entire theory of contracts. It comprises

a suitable refinement for services based on a *strong subcontract relation*, the proof that this refinement can be applied on each of the service inside a composition independently, and an effective procedure that can be used to prove whether a contract is a subcontract of another one.

We now discuss the related literature, first considering our papers (Bravetti and Zavattaro 2007a; Bravetti and Zavattaro 2007b), then moving to papers of other authors.

The theory of contracts reported in this paper is different from the theory reported in our previous paper (Bravetti and Zavattaro 2007a) in several aspects. The calculus considered in (Bravetti and Zavattaro 2007a) imposes a limitation to output actions that must always preceded by τ internal actions. On the contrary, in this paper we consider the standard output prefix. In this paper we have added locations to services, as well as to outputs in order to indicate the target of an invocation; this reflects more faithfully the Web Services technology in which invocations include both the address of the service and the operation to be invoked. As a technical consequence of adding location to the theory of (Bravetti and Zavattaro 2007a), we also have that the obtained subcontract relation is such that new inputs can be added in refined contracts, e.g. we have $a+b \preceq a$, even if we do not restrict the possible contexts to not perform output on those inputs (Proposition 4.5). In (Bravetti and Zavattaro 2007a) this, instead, does not hold: e.g. $[a] \parallel [\bar{a}.b] \parallel [\bar{b}]$ is a correct composition while $[a+b] \parallel [\bar{a}.b] \parallel [\bar{b}]$ is not, because the first and the second contracts are now in competition to consume the unique output on b (and if the first one grabs it then the second one will never terminate successfully). Moreover, in this paper we consider a stronger notion of compliance that requires to completely revisit the notion of subcontract. The most interesting result is that, even if the calculus in this paper is a more general language than (Bravetti and Zavattaro 2007a), we have achieved even stronger results thanks to the new notion of strong compliance. In particular, the strong subcontract relation of this paper can be defined abstracting away from both the input and the output alphabets of the services in the context (while this was not the case in (Bravetti and Zavattaro 2007a) where only inputs could be abstracted away). Moreover, the characterization of the strong subcontract relation (achieved resorting to the theory of should testing) is, from one hand, much more involved in this paper with respect to the characterization of the subcontract relation in (Bravetti and Zavattaro 2007a) (achieved resorting to should testing as well), from the other hand it allows new inputs to be added in refinements more liberally, by requiring them (their type) to be new just in state where they are added and not in the whole term as in (Bravetti and Zavattaro 2007a).

In (Bravetti and Zavattaro 2007b) we have extended the theory of contracts of (Bravetti and Zavattaro 2007a) combining it with a 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. (Busi *et al.* 2005; Busi *et al.* 2006) and by Carbone et al. (Carbone *et al.* 2007). The notion of choreography conformance is in general used to check whether a service can play a specific role within a given choreography. In (Bravetti and Zavattaro 2007b) we present a basic choreography language, and we define conformance between that language and (a variant of) the service/contract language of (Bravetti and Zavattaro 2007a).

taro 2007a): we check conformance by projecting a choreography on the considered role, and then exploiting (an enhanced version of) the notion of refinement of (Bravetti and Zavattaro 2007a).

We start the comparison with papers of other authors, observing that there are some important differences between our form of testing and the traditional one proposed by De Nicola-Hennessy (DH84). The main 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 $a.1 + a.b.1$, for which there are no guarantees of successful completion in any possible context. Moreover, all output actions of both the tester and the tested process should be immediately receivable. Another 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 by Rensink-Vogler (Rensink and Vogler 2005). In fact, we do not impose that all computations must succeed, but that all computations can always be extended in order to reach success.

We conclude our analysis of related work considering the theory of contracts by Fournet et al. (Fournet *et al.* 2004) and the one proposed by Carpineti et al. (Carpineti *et al.* 2006) and extended in (Laneve and Padovani 2007) and (Castagna *et al.* 2008).

In (Fournet *et al.* 2004) 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 stuckness. In (Fournet *et al.* 2004) 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 (Fournet *et al.* 2004). Another difference follows from the exploitation of the restriction $(\nu\tilde{a})$; this is used in (Fournet *et al.* 2004) to explicitly indicate the local channels of communication used between the contract C and the process Q . In our context we can make a stronger *closed-world* assumption (corresponding to a restriction on all channel names) because service contracts do not describe the entire behaviour of a service, but the flow of execution of its operations inside one session of communication.

The closed-world assumption is considered also by Carpineti et al. in (Carpineti *et al.* 2006) where, as in our case, a service oriented scenario is considered. In particular, in (Carpineti *et al.* 2006) a theory of contracts is defined for investigating the compatibility between one client and one service. Our paper consider 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 (Carpineti *et al.* 2006) where it is not in general possible to substitute a service exposing one contract with another one exposing a subcontract.

The work of Carpineti et al. has been extended in two ways, in (Laneve and Padovani 2007) by explicitly associating to a contract the considered input/output alphabet,

in (Castagna *et al.* 2008) by associating to services a dynamic filter which eliminates from the service behaviour those interactions that are not admitted by the considered contract. The explicit information about the input/output alphabet used in (Laneve and Padovani 2007) allows the corresponding theory of contracts to be applied also in multi-party compositions, but the independent refinement that we advocate can be only partially achieved. In fact, in general, a subcontract might consider a larger input/output alphabet with respect to an initial contract, but the input/output names added by each subcontract in the system must have empty intersection. In other terms, the selection of one subcontract with a larger input/output alphabet can influence the choice of the possible other subcontracts in the system. The dynamic filters of (Castagna *et al.* 2008) allow for independent refinement, at the price of synthesizing a specific filter for each service that eliminates the additional behaviours introduced by services exposing also a subcontract. Even if very interesting from a theoretical point of view, the practical application of filters is not yet clear. In fact, in general 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 it in multi-party systems composed of services running on different hosts.

Finally, also the work on type systems (e.g. that in (Kobayashi 2000) and (Carbone *et al.* 2006)) gives rise to notions of refinement in, somehow restricted, scenarios that can be syntactically characterized. For instance, the work in (Kobayashi 2000) 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 (Carbone *et al.* 2006) a subterm can be replaced by another one where inputs can be syntactically added in external choices and outputs can be syntactically removed from 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.

To conclude we comment on the issue of getting a complete characterization of the strong subcontract relation. Related contracts which are not captured by the sound characterization we present here include contracts which are not compliant with any other set of contracts as, e.g., $\mathbf{0}$ and $a.1 + a.b.1$ (see the above comparison with testing). Such contracts are all equivalent (one is the refinement of any of the others) and are often called contracts of “uncontrollable” services (i.e. there is no other service that, by controlling them, may lead to success). So the problem of getting a complete characterization includes the problem of establishing controllability of a service (see, e.g., (Lohmann 2008; Weinberg 2008)).

Acknowledgements

We thank the anonymous reviewers for their useful remarks and suggestions. The research reported in this paper was partially funded by EU Integrated Project Sensoria, contract n. 016004.

References

- Mario Bravetti and Gianluigi Zavattaro. Contract based Multi-party Service Composition. In *FSEN'07*, volume 4767 of LNCS, pages 207–222, 2007.
- 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.
- 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, 2005.
- 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, 2006.
- Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured Communication-Centred Programming for Web Services. In *ESOP'07*, volume 4421 of LNCS, pages 2–17. Springer, 2007.
- 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/cdllpaper/workingnote.pdf>
- 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, 2006.
- Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A Theory of Contracts for Web Services. In *POPL'08*, pages 261–272. ACM Press, 2008.
- Rocco De Nicola and Matthew Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, volume 34: 83–133, 1984.
- Cédric Fournet, C. A. R. Hoare, S. K. Rajamani, and Jakob Rehof. Stuck-Free Conformance. In *CAV'04*, volume 3114 of LNCS, pages 242–254, 2004.
- 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.
- 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, 2007.
- Frank Leymann. Web Services Flow Language (wsfl 1.0). Technical report, IBM Software Group, 2001.
- Niels Lohmann. Why does my service have no partners? In *WS-FM'08*, LNCS, to appear.
- Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
- Arend Rensink and Walter Vogler. Fair testing. *Information and Computation*, volume 205: 125–198. Elsevier, 2007.
- OASIS. *WS-BPEL: Web Services Business Process Execution Language Version 2.0*. Technical report, OASIS, 2003.
- Satish Thatte. XLANG: Web services for business process design. Technical report, Microsoft Corporation, 2001.

Daniela Weinberg. Efficient Controllability Analysis of Open Nets. In *WS-FM'08*, LNCS, to appear.

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

Appendix A. Process algebraic representation of contracts

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

Definition A.2. Let $\mathcal{T} = (S, \mathcal{L}, \longrightarrow, s_h, s_0)$ be a contract. A contract term $C \in \mathcal{P}_{con}$ is obtained from \mathcal{T} as follows.

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

Definition A.3. A homomorphism from a finite connected LTS with finite states $\mathcal{T} = (S, \mathcal{L}, \longrightarrow, s_h, s_0)$ to a finite connected LTS with finite states $\mathcal{T}' = (S', \mathcal{L}', \longrightarrow', s'_h, s'_0)$ is a function f from S to S' such that: $f(s_0) = s'_0$, $f(s_h) = s'_h$, and for all $s \in S$ we have $\{(\lambda, s') \mid f(s) \xrightarrow{\lambda} s'\} = \{(\lambda, f(s')) \mid s \xrightarrow{\lambda} s'\}$, i.e. the set of transitions performable by $f(s)$ is the same as the set of transitions performable by s when f -images of the target states are considered.

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

Proposition A.4. Let $\mathcal{T} = (S, \mathcal{L}, \longrightarrow, s_h, s_0)$ be a contract and $C \in \mathcal{P}_{con}$ be a contract term obtained from \mathcal{T} . There exists a (surjective) homomorphism from the semantics of C to \mathcal{T} itself.

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

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

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

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

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

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

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

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

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

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

Appendix B. Proof of Theorem 4.15

B.1. Technical Machinery Used in the Proof of the Theorem

We now introduce the technical machinery used in the proof of Theorem 4.15.

First, similarly as we did for contracts, we need to introduce a trasformation for systems $P \in \mathcal{P}_{conpar}$ that are to be executed in parallel with a given $[C]_l$, i.e. a contract C located at l , where both C and l are parameters of the transformation.

The semantics $\llbracket P \rrbracket$ of a system P is defined as being the labeled transition system LTS $\llbracket P \rrbracket = (S, \mathcal{A}_\tau, \longrightarrow, P)$, where S is the set of states reachable from P according to the transition relation defined by the operational rules for systems in Tables 1 and 2 and \longrightarrow is the subset of such a transition relation obtained by just considering transitions between states in S .

We transform $\llbracket P \rrbracket = (S, \mathcal{A}_\tau, \longrightarrow, P)$ into the LTS $(S', \mathcal{A}_\tau \cup \{\bar{a} \mid a \in \mathcal{N}\}, \longrightarrow', P)$ obtained as follows:

$$\begin{aligned}
& \text{--- } S' = S \cup \{new, newacc\} \\
& \text{--- } \longrightarrow' = \{(s, \lambda, s') \mid s \xrightarrow{\lambda} s' \wedge \lambda \neq \sqrt{\wedge} \exists a : \lambda = \bar{a}_l\} \\
& \quad \cup \{(s, \bar{a}, s') \mid s \xrightarrow{\bar{a}_l} s'\} \\
& \quad \cup \{(s, a_l, new) \mid a_l \in I(P) \wedge s \xrightarrow{a_l} \cdot\} \\
& \quad \cup \{(s, \tau, new) \mid \mathbf{nsO}(s) - I([C]_l) \neq \emptyset\} \\
& \quad \cup \{(s, \checkmark, newacc) \mid \exists s' : s \xrightarrow{\checkmark} s'\} \\
& \quad \cup \{(newacc, \checkmark', new)\}
\end{aligned}$$

The normal form for a system P (denoted with $\mathcal{NF}_{l,C}(P)$) is defined (similarly as for contracts) as follows, by using the operator $rec_X \theta$

$$\mathcal{NF}_{l,C}(P) = rec_{X_P} \theta$$

where θ is the set of S' -indexed equations

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

The semantics of $\mathcal{NF}_{l,C}(P)$ turns out to be the labeled transition system LTS $\llbracket \mathcal{NF}_{l,C}(P) \rrbracket = (S'', \mathcal{A}_\tau \cup \{\bar{a} \mid a \in \mathcal{N}\}, \longrightarrow'', \mathcal{NF}_{l,C}(P))$, where:

$$\text{--- } S'' = \{\mathcal{NF}_{l,C}(es) = \text{rec}_{X_{es}} \theta \mid es \in S'\}$$

$$\text{--- } \longrightarrow'' = \{(\mathcal{NF}_{l,C}(es), \lambda, \mathcal{NF}_{l,C}(es')) \mid es \xrightarrow{\lambda} es'\}$$

In conclusion, given a system P , we define $\text{map}_{l,C}(\llbracket P \rrbracket)$ to be the labeled transition system LTS $\llbracket \mathcal{NF}_{l,C}(P) \rrbracket$ defined above.

In order to build a test for the transformation $\mathcal{NF}(C)$ of a contract C located at l we have to consider $\overline{\mathcal{NF}_{l,C}(P)}$, i.e. the term $\text{rec}_{X_P} \theta'$ where θ' is obtained from θ in $\mathcal{NF}_{l,C}(P) = \text{rec}_{X_P} \theta$ by turning every $a_{l'}$ occurring in θ , for any a, l' , into $\bar{a}_{l'}$ and every $\bar{a}_{l'}$ occurring in θ , for any a, l' , into $a_{l'}$ (a , respectively). The LTS $\llbracket \overline{\mathcal{NF}_{l,C}(P)} \rrbracket = (S''', \mathcal{A}_\tau, \longrightarrow''', \overline{\mathcal{NF}_{l,C}(P)})$ turns out to be a transformation of $\llbracket \mathcal{NF}_{l,C}(P) \rrbracket$ where input/output actions inside states are inverted as described above and whose transition labels are transformed in the same way.

We now introduce mapping of traces of $\llbracket C \rrbracket$ into $\text{map}(\llbracket C \rrbracket) = \llbracket \mathcal{NF}(C) \rrbracket$ and mapping of traces of $\llbracket P \rrbracket$ into $\text{map}_{l,C}(\llbracket P \rrbracket) = \llbracket \mathcal{NF}_{l,C}(P) \rrbracket$. 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. $tr = s_0 \xrightarrow{\lambda_1} s_1 \xrightarrow{\lambda_2} \dots \xrightarrow{\lambda_{n-1}} s_{n-1} \xrightarrow{\lambda_n} s_n$. We denote concatenation of two traces $tr' \in Tr_n^{\mathcal{T}}$ and $tr'' \in Tr_m^{\mathcal{T}}$ such that $s'_n = s''_0$ by $tr' \widehat{\ } tr''$ defined as the trace $tr \in Tr_{n+m}^{\mathcal{T}}$ with $s_i = s'_i \forall 0 \leq i \leq n$, $\lambda_i = \lambda'_i \forall 1 \leq i \leq n$, $s_{n+i} = s''_i \forall 1 \leq i \leq m$ and $\lambda_{n+i} = \lambda''_i \forall 1 \leq i \leq m$. We also use $\text{less}_i(tr)$ 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 $\text{less}(tr)$ to stand for $\text{less}_1(tr)$. Finally we denote with $\text{vis}(tr)$ the sequence of visible labels of the trace tr , i.e., the function λ' from an interval $[1, m]$ of integers to non- τ labels defined by induction on the length $n \geq 0$ of trace tr as follows: if $n = 0$ then $\text{vis}(tr) = \emptyset$, if $n \geq 1$ then $\text{vis}(tr)$ is the sequence of labels λ'' obtained from $\lambda' = \text{vis}(\text{less}(tr))$, with $\text{dom}(\lambda') = [1, m]$, as follows. If $\lambda_n = \tau$ then simply $\lambda'' = \lambda'$, otherwise $\text{dom}(\lambda'') = [1, m+1]$ and $\lambda''(i) = \lambda'(i) \forall 1 \leq i \leq m$ and $\lambda''(m+1) = \lambda_n$.

Let us consider a transition $s \xrightarrow{\lambda} s'$ of $\llbracket C \rrbracket$. We can take $tr = s \xrightarrow{\lambda} s'$ with $tr \in Tr_1^{\llbracket C \rrbracket}$. We define $\text{map}(tr)$ as follows. If $\exists a, l' : \lambda = \bar{a}_{l'}$ then $\text{map}(tr) = \mathcal{NF}(s) \xrightarrow{\lambda} \mathcal{NF}(s')$. Otherwise, if $\lambda = \bar{a}_{l'}$ then $\text{map}(tr) = \mathcal{NF}(s) \xrightarrow{\tau} \mathcal{NF}((s, \bar{a}_{l'}, s')) \xrightarrow{\bar{a}_{l'}} \mathcal{NF}(s')$. We then define the mapping $\text{map}(tr)$ of a whatever transition $tr = (s, \lambda) \in Tr_n^{\llbracket C \rrbracket}$ to be the transition $tr' = (s', \lambda') \in Tr_m^{\llbracket \mathcal{NF}(C) \rrbracket}$ with $s'_0 = \mathcal{NF}(s_0)$ and $s'_m = \mathcal{NF}(s_n)$ achieved by induction on $n \geq 0$ as follows. If $n = 0$ then $\text{map}(tr)$ is the trace $tr' \in Tr_0^{\llbracket \mathcal{NF}(C) \rrbracket}$ such that $s'_0 = \mathcal{NF}(s_0)$. If $n \geq 1$ then $\text{map}(tr) = \text{map}(\text{less}(tr)) \widehat{\ } \text{map}(s_{n-1} \xrightarrow{\lambda_n} s_n)$. It is immediate to verify that for any $tr \in Tr^{\llbracket C \rrbracket}$, $\text{vis}(\text{map}(tr)) = \text{vis}(tr)$. Moreover we have that $\text{map} : Tr^{\llbracket C \rrbracket} \rightarrow Tr^{\llbracket \mathcal{NF}(C) \rrbracket}$ is injective (because the last transition of the trace singles out at each inductive step a unique mapping that can produce it) and is surjective

over the codomain of traces that begin and end with a state of the form $\mathcal{NF}(s)$ with $s \in \llbracket C \rrbracket$ (due to the way $map(\llbracket C \rrbracket)$ is defined).

Let us consider a transition $s \xrightarrow{\lambda} s'$ of $\llbracket P \rrbracket$. We can take $tr = s \xrightarrow{\lambda} s'$ with $tr \in Tr_1^{\llbracket P \rrbracket}$. We define $map_l(tr)$ as follows. If $\lambda \neq \surd \wedge \exists a : \lambda = \bar{a}_l$ then $map_l(tr) = \mathcal{NF}_{l,C}(s) \xrightarrow{\lambda} \mathcal{NF}_{l,C}(s')$. Otherwise, if $\lambda = \surd$ then $map_l(tr) = \mathcal{NF}_{l,C}(s) \xrightarrow{\surd} \mathcal{NF}_{l,C}(newacc)$; if, instead, $\lambda = \bar{a}_l$ then $map_l(tr) = \mathcal{NF}_{l,C}(s) \xrightarrow{\bar{a}} \mathcal{NF}_{l,C}(s')$. We then define the mapping $map_l(tr)$ of a whatever transition $tr = (s, \lambda) \in Tr_n^{\llbracket P \rrbracket}$ to be the transition $tr' = (s', \lambda') \in Tr_n^{\llbracket \mathcal{NF}(P) \rrbracket}$ with $s'_0 = \mathcal{NF}_{l,C}(s_0)$ and, in the case $n \geq 1$, $s'_n = \mathcal{NF}_{l,C}(s_n)$ if $\lambda_n \neq \surd$, $s'_n = \mathcal{NF}_{l,C}(newacc)$ otherwise, achieved by induction on $n \geq 0$ as follows. If $n = 0$ then $map_l(tr)$ is the trace $tr' \in Tr_0^{\llbracket \mathcal{NF}_{l,C}(P) \rrbracket}$ such that $s'_0 = \mathcal{NF}_{l,C}(s_0)$. If $n \geq 1$ then $map_l(tr) = map_l(less(tr)) \hat{\ } map_l(s_{n-1} \xrightarrow{\lambda_n} s_n)$. It is immediate to verify that for any $tr \in Tr^{\llbracket P \rrbracket}$, $vis(map_l(tr)) = vis(tr)\{\bar{a}/\bar{a}_l \mid a \in \mathcal{N}\}$ denoting replacement of any label \bar{a}_l occurring in the sequence of visible labels with \bar{a} . Moreover we have that $map_l : Tr^{\llbracket P \rrbracket} \rightarrow Tr^{\llbracket \mathcal{NF}_{l,C}(P) \rrbracket}$ is injective (because the last transition of the trace singles out at each inductive step a unique mapping that can produce it and because in P there cannot be states with multiple outgoing \surd transitions) and is surjective over the codomain of traces that begin with a state of the form $\mathcal{NF}_{l,C}(s)$ with $s \in \llbracket P \rrbracket$ and end with a state of the form $\mathcal{NF}_{l,C}(s)$ with $s \in \llbracket P \rrbracket$ or $\mathcal{NF}_{l,C}(newacc)$.

We finally define some other auxiliary functions that will be used in the proof. We first define a function that removes square brackets and attached location from states. Given a state $s \in \llbracket [C]_l \rrbracket$, we define $rm_l(s)$ to be $s' \in [C]$ such that $s = [s']_l$. Moreover given a trace $tr \in Tr^{\llbracket C \rrbracket}$, we define $[tr]_l$ to be $tr' = (s', \lambda')$ defined as: $s'_i = [s_i]_l \forall i$ and for each i we have $\lambda'_i = a_l$ if $\lambda_i = a$ for some $a \in \mathcal{N}$; $\lambda'_i = \lambda_i$ otherwise. Therefore, similarly as for states, given a trace $tr \in Tr^{\llbracket [C]_l \rrbracket}$, we define $rm_l(tr)$ to be $tr' \in Tr^{\llbracket C \rrbracket}$ such that $tr = [tr']_l$. Finally, given a trace $tr \in Tr^{\llbracket \mathcal{NF}_{l,C}(P) \rrbracket}$ or $tr \in Tr^{\llbracket \mathcal{NF}_{l,C}(P) \rrbracket}$, we define \bar{tr} to be $tr' = (s', \lambda')$ defined as: $s'_i = \bar{s}_i \forall i$ and $\lambda'_i = \bar{\lambda}_i \forall i$ (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 $vis(\bar{tr}) = \overline{vis(tr)}$ denoting the application of the overbar to any label occurring in the sequence of visible labels.

B.2. Proof of the Theorem

We are now ready to report the proof of Theorem 4.15.

According to the definition of should-testing of (Rensink and Vogler 2005), since $\mathcal{NF}(C' \setminus (I(C') - I(C))) \preceq_{test} \mathcal{NF}(C)$ we have that, for every test t , if $\mathcal{NF}(C) \mathbf{shd} t$, then also $\mathcal{NF}(C' \setminus (I(C') - I(C))) \mathbf{shd} t$.

Let us now consider $l \in Loc$, $l \notin oloc(C) \cup oloc(C')$, and $P \in \mathcal{P}_{compar}$, $l \notin loc(P)$, such that $([C]_l \parallel P) \Downarrow$. In the following we will provide a first subproof that this implies $\mathcal{NF}(C) \mathbf{shd} \overline{\mathcal{NF}_{l,C}(P)}$. Since $\mathcal{NF}(C' \setminus (I(C') - I(C))) \preceq_{test} \mathcal{NF}(C)$, from this we can derive that also $\mathcal{NF}(C' \setminus (I(C') - I(C))) \mathbf{shd} \overline{\mathcal{NF}_{l,C}(P)}$. In the following we will provide a second subproof that this implies $([C' \setminus (I(C') - I(C))]_l \parallel P) \Downarrow$. The thesis, then, directly follows from Lemma 4.14.

Before presenting the two subproofs we explain some convention about the notation

that we will use in such proofs, so to help the reader. In general we will deal with “global” traces of terms that are a parallel between (the transformation of) a contract and (the transformation of) a system representing its context. Such traces will be denoted by $tr = (s, \lambda), tr' = (s', \lambda'), tr'' = (s'', \lambda''), \dots$. When we will consider “local” traces of contracts and systems that are included in such “global” traces we will denote them by $\dot{tr} = (\dot{s}, \dot{\lambda}), \dot{tr}' = (\dot{s}', \dot{\lambda}'), \dot{tr}'' = (\dot{s}'', \dot{\lambda}''), \dots$, respectively, for (transformation of) contracts and by $\ddot{tr} = (\ddot{s}, \ddot{\lambda}), \ddot{tr}' = (\ddot{s}', \ddot{\lambda}'), \ddot{tr}'' = (\ddot{s}'', \ddot{\lambda}''), \dots$, respectively, for (transformation of) systems.

First subproof: $([C]_l \parallel P) \Downarrow \Rightarrow \mathcal{NF}(C) \text{ shd } \overline{\mathcal{NF}_{l,C}(P)}$.

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

We have three cases for the structure of the last state of trace \dot{tr} :

- 1 $\dot{s}_n = \mathcal{NF}(s)$ for some $s \in \llbracket C \rrbracket$
- 2 $\dot{s}_n = \mathcal{NF}(s, \bar{a}_{l'}, s')$ for some $s, s' \in \llbracket C \rrbracket, a \in \mathcal{N}$ and $l' \in Loc$
- 3 $\dot{s}_n = \mathcal{NF}(new)$

Moreover, we have three cases for the structure of the last state of trace \ddot{tr} :

- 1 $\ddot{s}_m = \overline{\mathcal{NF}_{l,C}(s)}$ for some $s \in \llbracket P \rrbracket$
- 2 $\ddot{s}_m = \overline{\mathcal{NF}_{l,C}(newacc)}$
- 3 $\ddot{s}_m = \overline{\mathcal{NF}_{l,C}(new)}$

We first show that, whatever the case for \ddot{tr} , the case (3) for \dot{tr} cannot be obtained, by contradiction. In this case, we should have (by definition of $map(\llbracket C \rrbracket)$) that $\lambda_n = a$ for some $a \in I(C)$. Let us consider the trace $\dot{tr}' = [map^{-1}(less(\dot{tr}))]_l \in ITr_n^{\llbracket [C]_l \rrbracket}$. We must have, by def. of $map(\llbracket C \rrbracket)$, that $\dot{s}'_n \xrightarrow{a_l}$.

Moreover, it must also be that $\ddot{\lambda}_{m-i} = a$ for some $i \geq 0$ such that, if $i \geq 1$, $\ddot{\lambda}_{(m-i)+j} = \tau$ for $1 \leq j \leq i$. Let us consider $\ddot{tr}' = map_l^{-1}(\overline{less_{i+1}(\ddot{tr})}) \in ITr_{m-i}^{\llbracket P \rrbracket}$. We must have, by def. of $map_{l,C}(\llbracket P \rrbracket)$, that $\ddot{s}'_{m-i} \xrightarrow{\bar{a}_l}$. Therefore $\bar{a}_l \in \text{nsO}(P)$.

Since $vis(\dot{tr}') = vis(less(\dot{tr}))\{a_l/a \mid a \in \mathcal{N}\}$, $vis(\ddot{tr}') = \overline{vis(less_i(\ddot{tr}))}\{\bar{a}_l/\bar{a} \mid a \in \mathcal{N}\}$ and also $vis(less(\dot{tr})) = vis(less_i(\ddot{tr}))$, we have $vis(\dot{tr}') = vis(\ddot{tr}')$. Therefore, there exists $tr' \in ITr_r^{\llbracket [C]_l \parallel P \rrbracket}$, with $\lambda'_i = \tau \forall i$, such that $s'_r = \dot{s}'_n \parallel \ddot{s}'_{m-i}$ and $\neg \text{exceptionFree}(s'_r)$.

We now show that, the case (3) for \ddot{tr} cannot be obtained, by contradiction. In this case we would have (by definition of $map_{l,C}(\llbracket P \rrbracket)$) three cases:

- (a) $\ddot{\lambda}_m = \bar{a}_{l'}$, for some a, l' with $a_{l'} \in I(P)$
- (b) $\ddot{\lambda}_m = \tau$
- (c) $\ddot{\lambda}_m = \surd'$

— In the case (a) we do the following. Let us consider $\ddot{tr}' = map_l^{-1}(\overline{less(\ddot{tr})}) \in ITr_{m-1}^{\llbracket P \rrbracket}$.

We must have, by def. of $map_{l,C}(\llbracket P \rrbracket)$, that $\ddot{s}'_{m-1} \xrightarrow{\bar{a}_{l'}}$.

Moreover, it must also be that $\ddot{\lambda}_{n-i} = \bar{a}_{l'}$ for some $i \geq 0$ such that, if $i \geq 1$, $\ddot{\lambda}_{(n-i)+j} = \tau$ for $1 \leq j \leq i$. Let us consider $\dot{tr}' = [map^{-1}(less_{i+2}(\ddot{tr}))]_l \in ITr_n^{\llbracket [C]_l \rrbracket}$.

We must have, by def. of $map(\llbracket C \rrbracket)$, that $\dot{s}'_n \xrightarrow{\bar{a}_{l'}}$. Therefore $\bar{a}_{l'} \in \text{nsO}([C]_l)$.

Since $vis(\dot{tr}') = vis(less_i(\ddot{tr}))\{a_l/a \mid a \in \mathcal{N}\}$, $vis(\ddot{tr}') = \overline{vis(less(\ddot{tr}))}\{\bar{a}_l/\bar{a} \mid a \in \mathcal{N}\}$

and $\text{vis}(\text{less}_i(\dot{t}r)) = \text{vis}(\text{less}(\dot{t}r))$, we have $\text{vis}(\dot{t}r') = \overline{\text{vis}(\dot{t}r')}$. Therefore, there exists $\dot{t}r' \in \text{ITr}_{r'}^{\llbracket [C]_l \rrbracket P}$, with $\lambda'_i = \tau \forall i$, such that $s'_{r'} = \dot{s}'_{n'} \parallel \dot{s}'_{m-1}$ and $\neg \text{exceptionFree}(s'_{r'})$.

- In the case (b) we do the following. Let us consider $\dot{t}r' = \overline{\text{map}_l^{-1}(\text{less}(\dot{t}r))} \in \text{ITr}_{m-1}^{\llbracket P \rrbracket}$. We must have, by def. of $\text{map}_{l,C}(\llbracket P \rrbracket)$, that $\text{nso}(\dot{s}'_{m-1}) - \overline{I([C]_l)} \neq \emptyset$. Now we have two possible cases for $\dot{t}r$: case (1) and case (2) above. In case (1) we consider $\dot{t}r' = [\text{map}^{-1}(\dot{t}r)]_l \in \text{ITr}_{n'}^{\llbracket [C]_l \rrbracket}$. In case (2) we consider $\dot{t}r' = [\text{map}^{-1}(\text{less}(\dot{t}r))]_l \in \text{ITr}_{n'}^{\llbracket [C]_l \rrbracket}$ and we have $\text{vis}(\dot{t}r) = \text{vis}(\text{less}(\dot{t}r))\{a_l/a \mid a \in \mathcal{N}\}$. Hence in both cases $\text{vis}(\dot{t}r') = \text{vis}(\dot{t}r)\{a_l/a \mid a \in \mathcal{N}\}$. Since $\text{vis}(\dot{t}r') = \text{vis}(\text{less}(\dot{t}r'))\{\bar{a}_l/\bar{a} \mid a \in \mathcal{N}\} = \overline{\text{vis}(\dot{t}r)}\{\bar{a}_l/\bar{a} \mid a \in \mathcal{N}\}$, we have $\text{vis}(\dot{t}r') = \overline{\text{vis}(\dot{t}r')}$. Therefore, there exists $\dot{t}r' \in \text{ITr}_{r'}^{\llbracket [C]_l \rrbracket P}$, with $\lambda'_i = \tau \forall i$, such that $s'_{r'} = \dot{s}'_{n'} \parallel \dot{s}'_{m-1}$ and $\neg \text{exceptionFree}(s'_{r'})$.
- The case (c) contradicts the fact that the initial trace $\dot{t}r$ has the following property: $\lambda_i \neq \sqrt{\forall i}$.

If $\dot{t}r$ is as in case (2), from $\dot{s}_m = \overline{\mathcal{NF}_{l,C}(\text{newacc})}$ we derive (by definition of $\text{map}_{l,C}(\llbracket P \rrbracket)$) that $\dot{s}_m \xrightarrow{\sqrt{\prime}}$, so we immediately conclude that $s_r = \dot{s}_n \parallel_{\mathcal{A}} \dot{s}_m \xrightarrow{\sqrt{\prime}}$ and we are done.

Therefore we have only case (1) left for $\dot{t}r$. Let us consider $\dot{t}r' = \text{map}_l^{-1}(\overline{\dot{t}r}) \in \text{ITr}_m^{\llbracket P \rrbracket}$. We have $\dot{s}_m = \overline{\mathcal{NF}_{l,C}(\dot{s}'_m)}$.

We start by taking $\dot{t}r$ to be as in case (1). We consider $\dot{t}r' = [\text{map}^{-1}(\dot{t}r)]_l \in \text{ITr}_{n'}^{\llbracket [C]_l \rrbracket}$. We have $\dot{s}_n = \mathcal{NF}(rm_l(\dot{s}'_{n'}))$. Moreover, $\text{vis}(\dot{t}r') = \overline{\text{vis}(\dot{t}r)}\{\bar{a}_l/\bar{a} \mid a \in \mathcal{N}\} = \overline{\text{vis}(\dot{t}r)}\{a_l/a \mid a \in \mathcal{N}\} = \text{vis}(\dot{t}r')$.

Therefore, there exists $\dot{t}r' \in \text{ITr}_{r'}^{\llbracket [C]_l \rrbracket P}$, with $\lambda'_i = \tau \forall i$, such that $s'_{r'} = \dot{s}'_{n'} \parallel \dot{s}'_m$.

Since $([C]_l \parallel P) \Downarrow$ we have that there exists $\dot{t}r'' \in \text{Tr}_{r''}^{\llbracket [C]_l \rrbracket P}$ with $s''_0 = s'_{r'}$, $\lambda''_i = \tau \forall 1 \leq i \leq r'' - 1$ and $\lambda''_{r''} = \sqrt{\prime}$. Therefore, there exist $\dot{t}r'' \in \text{Tr}_{n''}^{\llbracket [C]_l \rrbracket}$ with $\dot{s}''_0 = \dot{s}'_{n'}$, and $\dot{t}r'' \in \text{Tr}_{m''}^{\llbracket P \rrbracket}$, with $\dot{s}''_0 = \dot{s}'_m$ corresponding to the local moves performed by the two parallel processes when doing trace $\dot{t}r''$, such that $\text{vis}(\dot{t}r'') = \overline{\text{vis}(\dot{t}r'')}$.

Let us now consider $\dot{t}r''' = \text{map}(rm_l(\dot{t}r'')) \in \text{Tr}_{n'''}^{\llbracket \mathcal{NF}(C) \rrbracket}$. We have $\dot{s}'''_0 = \mathcal{NF}(rm_l(\dot{s}''_0)) = \mathcal{NF}(rm_l(\dot{s}'_{n'}))$ and $\dot{\lambda}'''_{n'''} = \sqrt{\prime}$. Let us also consider $\dot{t}r''' = \text{map}_l(\dot{t}r'') \in \text{Tr}_{m'''}^{\llbracket \mathcal{NF}_{l,C}(P) \rrbracket}$. We have $\dot{s}'''_0 = \overline{\mathcal{NF}_{l,C}(\dot{s}''_0)} = \overline{\mathcal{NF}_{l,C}(\dot{s}'_m)}$ and $\dot{\lambda}'''_{m'''} = \sqrt{\prime}$. Moreover, $\text{vis}(\dot{t}r''') = \text{vis}(\dot{t}r'')\{a_l/a \mid a \in \mathcal{N}\} = \text{vis}(\dot{t}r'')\{\bar{a}_l/\bar{a} \mid a \in \mathcal{N}\} = \text{vis}(\dot{t}r''')$.

Therefore, there exists $\dot{t}r''' \in \text{Tr}_{r'''}^{\llbracket \mathcal{NF}(C) \rrbracket_{\mathcal{A}} \overline{\mathcal{NF}_{l,C}(P)}}$. Such trace has as its initial state $\dot{s}'''_0 = \mathcal{NF}(rm_l(\dot{s}'_{n'})) \parallel_{\mathcal{A}} \overline{\mathcal{NF}_{l,C}(\dot{s}'_m)} = \dot{s}_n \parallel_{\mathcal{A}} \dot{s}_m = s_r$, and $s'_{r'''} = \dot{s}'''_{n'''} \parallel_{\mathcal{A}} \dot{s}'''_{m'''}$. Moreover, since, by def. of $\text{map}_{l,C}(\llbracket P \rrbracket)$, $\dot{\lambda}'''_{m'''} = \sqrt{\prime} \Rightarrow \dot{s}'''_{m'''} = \overline{\mathcal{NF}_{l,C}(\text{newacc})} \xrightarrow{\sqrt{\prime}}$, we have $s'_{r'''} \xrightarrow{\sqrt{\prime}}$.

Finally, we take $\dot{t}r$ to be as in case (2). We consider $\dot{t}r' = [\text{map}^{-1}(\text{less}(\dot{t}r))]_l \in \text{ITr}_{n'}^{\llbracket [C]_l \rrbracket}$. We have that $\dot{s}_{n-1} = \mathcal{NF}(rm_l(\dot{s}'_{n'}))$ and, by def. of $\text{map}(\llbracket C \rrbracket)$, there exist \hat{s} such that $\dot{s}_n = \overline{\mathcal{NF}(rm_l(\dot{s}'_{n'}), \bar{a}_l, rm_l(\hat{s}))}$ and $\dot{\lambda}_n = \tau$. Moreover, we have that $\text{vis}(\dot{t}r') = \text{vis}(\dot{t}r)\{\bar{a}_l/\bar{a} \mid a \in \mathcal{N}\} = \text{vis}(\dot{t}r)\{a_l/a \mid a \in \mathcal{N}\} = \text{vis}(\text{less}(\dot{t}r))\{a_l/a \mid a \in \mathcal{N}\} = \text{vis}(\dot{t}r')$.

Therefore, there exists $\dot{t}r' \in \text{ITr}_{r'}^{\llbracket [C]_l \rrbracket P}$, with $\lambda'_i = \tau \forall i$, such that $s'_{r'} = \dot{s}'_{n'} \parallel \dot{s}'_m$.

Since $([C]_l \parallel P) \Downarrow$, we have $\text{exceptionFree}(s'_{r'})$. Therefore, since, by def. of $\text{map}(\llbracket C \rrbracket)$, $\dot{s}'_{n'} \xrightarrow{\bar{a}_l} \hat{s}$, hence $\bar{a}_l \in \text{nso}([C]_l)$, we must have that there exist \check{s} such that $\dot{s}'_m \xrightarrow{a_l} \check{s}$.

Since $([C]_l \parallel P) \Downarrow$, we have that there exists $\dot{t}r'' \in \text{Tr}_{r''}^{\llbracket [C]_l \rrbracket P}$ with $\dot{s}''_0 = \hat{s} \parallel \check{s}$, $\lambda''_i = \tau \forall 1 \leq i \leq r'' - 1$ and $\lambda''_{r''} = \sqrt{\prime}$.

Therefore, there exist $\dot{tr}'' \in Tr_{n''}^{[[C]_l]}$ with $\dot{s}_0'' = \hat{s}$ and $\ddot{tr}'' \in Tr_{m''}^{[P]}$, with $\dot{s}_0'' = \check{s}$ corresponding to the local moves performed by the two parallel processes when doing trace tr'' , such that $vis(\dot{tr}'') = vis(\ddot{tr}'')$.

Let us now consider $\dot{tr}''' = map(rm_l(\dot{tr}'')) \in Tr_{n'''}^{[N\mathcal{F}(C)]}$. We have $\dot{s}_0''' = N\mathcal{F}(rm_l(\dot{s}_0'')) = N\mathcal{F}(rm_l(\hat{s}))$ and $\dot{\lambda}_{n'''} = \checkmark$. Let us also consider $\ddot{tr}''' = \overline{map_l(\ddot{tr}'')} \in Tr_{m'''}^{[N\mathcal{F}_{l,C}(P)]}$. We have $\ddot{s}_0''' = \overline{N\mathcal{F}_{l,C}(\ddot{s}_0'')} = \overline{N\mathcal{F}_{l,C}(\check{s})}$ and $\ddot{\lambda}_{m'''} = \checkmark$. Moreover, $vis(\dot{tr}''') = vis(\ddot{tr}''')\{a_l/a \mid a \in \mathcal{N}\} = vis(\ddot{tr}'')\{a_l/a \mid a \in \mathcal{N}\} = vis(\dot{tr}'')\{\bar{a}_l/\bar{a} \mid a \in \mathcal{N}\} = vis(\ddot{tr}''')$.

Therefore, there exists $tr''' \in Tr_{r'''}^{[N\mathcal{F}(C)]_{\mathcal{A}N\mathcal{F}_{l,C}(P)}}$. Such trace has as its initial state $s_0''' = N\mathcal{F}(rm_l(\hat{s}))_{\mathcal{A}N\mathcal{F}_{l,C}(\check{s})}$, and $s_{r'''} = \dot{s}_{n'''}_{\mathcal{A}\ddot{s}_m'''}'$. Moreover, we have, by def. of $map([[C]])$, $\dot{s}_n = N\mathcal{F}(rm_l(\dot{s}'_n), \bar{a}_l, rm_l(\hat{s})) \xrightarrow{\bar{a}_l} N\mathcal{F}(rm_l(\hat{s}))$ and, since $\dot{s}'_m \xrightarrow{a_l} \check{s}$, we have, by def. of $map_{l,C}([P])$, $\ddot{s}_m = \overline{N\mathcal{F}_{l,C}(\ddot{s}'_m)} \xrightarrow{\bar{a}_l} \overline{N\mathcal{F}_{l,C}(\check{s})}$. Thus we can build the trace $s_r = \dot{s}_n_{\mathcal{A}\ddot{s}_m} \xrightarrow{\bar{a}_l} N\mathcal{F}(rm_l(\hat{s}))_{\mathcal{A}\overline{N\mathcal{F}_{l,C}(\check{s})}} \frown tr'''$. Moreover, since, by def. of $map_{l,C}([P])$, $\ddot{\lambda}_{m'''} = \checkmark \Rightarrow \ddot{s}_{m'''} = \overline{N\mathcal{F}_{l,C}(newacc)} \xrightarrow{\checkmark'}$, we have $s_{r'''} \xrightarrow{\checkmark'}$.

Second subproof: $N\mathcal{F}(\tilde{C}') \text{ shd } \overline{N\mathcal{F}_{l,C}(P)} \Rightarrow ([\tilde{C}']_l | P) \Downarrow$, with $\tilde{C}' = C' \Downarrow (I(C') - I(C))$. Note that $I(\tilde{C}') = I(C)$.

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

Let us now consider $\dot{tr}' = map(rm_l(\dot{tr})) \in ITr_{n'}^{[N\mathcal{F}(\tilde{C}')]}$.

We have $\dot{s}'_n = N\mathcal{F}(rm_l(\dot{s}_n))$. Let us also consider $\ddot{tr}' = \overline{map_l(\ddot{tr})} \in ITr_m^{[\overline{N\mathcal{F}_{l,C}(P)}]}$. We have $\ddot{s}'_m = \overline{N\mathcal{F}_{l,C}(\ddot{s}_m)}$.

Moreover, $vis(\dot{tr}') = vis(\dot{tr})\{a_l/a \mid a \in \mathcal{N}\} = \overline{vis(\ddot{tr})\{a_l/a \mid a \in \mathcal{N}\}} = \overline{vis(\ddot{tr})\{\bar{a}_l/\bar{a} \mid a \in \mathcal{N}\}} = vis(\ddot{tr}')$. Therefore, there exists $tr' \in ITr_{r'}^{[N\mathcal{F}(\tilde{C}')]_{\mathcal{A}\overline{N\mathcal{F}_{l,C}(P)}}$ with $s'_{r'} = \dot{s}'_{n'}_{\mathcal{A}\ddot{s}'_m}$ which is equal to $N\mathcal{F}(rm_l(\dot{s}_n))_{\mathcal{A}\overline{N\mathcal{F}_{l,C}(\ddot{s}_m)}}$.

We start by showing that $\text{exceptionFree}(s_r)$ holds.

We first observe that $\text{exceptionFree}(s_r = \dot{s}_n_{\mathcal{A}\ddot{s}_m}) \iff \mathbf{nso}(\dot{s}_n_{\mathcal{A}\ddot{s}_m}) = (\mathbf{nso}(\dot{s}_n) - \{\bar{a}_l \mid \dot{s}_m \xrightarrow{a_l}\}) \cup (\mathbf{nso}(\ddot{s}_m) - \{\bar{a}_l \mid \dot{s}_n \xrightarrow{a_l}\}) = \emptyset \iff \mathbf{nso}(\dot{s}_n) - \{\bar{a}_l \mid \dot{s}_m \xrightarrow{a_l}\} = \emptyset \wedge \mathbf{nso}(\ddot{s}_m) - \{\bar{a}_l \mid \dot{s}_n \xrightarrow{a_l}\} = \emptyset$. We show that both such statements hold in the following two items.

— We show that there is no $\bar{a}_l \in \mathbf{nso}(\dot{s}_n)$ such that $\dot{s}_m \not\xrightarrow{a_l}$, by contradiction. First of all, we have, by def. of \mathbf{nso} , that, since \dot{s}_n is a derivative of $[\tilde{C}']_l$, $\mathbf{nso}(\dot{s}_n) = \{\bar{a}_l \mid \dot{s}_n \xrightarrow{\bar{a}_l}\}$. Hence we consider any \bar{a}_l and \hat{s} such that $\dot{s}_n \xrightarrow{\bar{a}_l} \hat{s}$ and we show that it cannot be that $\dot{s}_m \not\xrightarrow{a_l}$. We first observe that, by def. of $map([\tilde{C}']_l)$, we have $\dot{s}'_{n'} = N\mathcal{F}(rm_l(\dot{s}_n)) \xrightarrow{\tau} (\dot{s}'_{n'}, \bar{a}_l, N\mathcal{F}(rm_l(\hat{s}))) \xrightarrow{\bar{a}_l} N\mathcal{F}(rm_l(\hat{s}))$. Now we have the following two cases depending on whether $a_l \in I(P)$ holds true or false.

- 1 If $a_l \in I(P)$, then from $\dot{s}_m \not\xrightarrow{a_l}$ we derive, by def. of $map_{l,C}([P])$, that $\ddot{s}'_m = \overline{N\mathcal{F}_{l,C}(\ddot{s}_m)} \xrightarrow{\bar{a}_l} \overline{N\mathcal{F}_{l,C}(new)}$. Therefore we can build the trace $tr' \frown s'_{r'} = \dot{s}'_{n'}_{\mathcal{A}\ddot{s}'_m} \xrightarrow{\tau} (\dot{s}'_{n'}, \bar{a}_l, N\mathcal{F}(rm_l(\hat{s})))_{\mathcal{A}\ddot{s}'_m} \xrightarrow{\bar{a}_l} N\mathcal{F}(rm_l(\hat{s}))_{\mathcal{A}\overline{N\mathcal{F}_{l,C}(new)}}$. For such a trace there cannot be a continuing trace

(a trace starting in the last state of the above trace) including a \surd' transition because, by def. of $\text{map}_{l,C}(\llbracket P \rrbracket)$, the state $\overline{\mathcal{NF}_{l,C}(\text{new})}$ of the test has no outgoing transitions.

- 2 If $a_{l'} \notin I(P)$, then we build the following trace $tr' \widehat{\ } s'_{r'}$, which is equal to $\dot{s}'_{n'} \parallel_{\mathcal{A}} \dot{s}'_m \xrightarrow{\tau} (\dot{s}'_{n'}, \bar{a}_{l'}, \mathcal{NF}(\text{rm}_l(\hat{s}))) \parallel_{\mathcal{A}} \dot{s}'_m$. For such a trace there cannot be a continuing trace (a trace starting in the last state of the above trace) including a \surd' transition for the following reason. By definition of $\text{map}(\llbracket \tilde{C}' \rrbracket)$ we have that $(\dot{s}'_{n'}, \bar{a}_{l'}, \mathcal{NF}(\text{rm}_l(\hat{s})))$ can only do a $\bar{a}_{l'}$ transition and, since $a_{l'} \notin I(P)$, any trace of the test starting in \dot{s}'_m cannot include a synchronizing $\bar{a}_{l'}$ transition. Therefore in any continuing trace the test cannot do a \surd transition (that requires synchronization), hence also it cannot do a \surd' transition because, by def. of $\text{map}_{l,C}(\llbracket P \rrbracket)$, \surd' transitions can only be done in the target state $\overline{\mathcal{NF}_{l,C}(\text{newacc})}$ of \surd transitions.

— We show that there is no $\bar{a}_{l'} \in \mathbf{ns}(\dot{s}_m)$ such that $\dot{s}_n \xrightarrow{a_{l'}} \cdot$, by contradiction. We have two cases depending on whether $l' = l$ and $a \in I(C) = I(\tilde{C}')$, i.e. $a_{l'} \in I([C]_l)$, is true or false.

- 1 If $l' = l$ and $a \in I(C) = I(\tilde{C}')$, then, from $\dot{s}_n \xrightarrow{a_{l'}} \cdot$ we derive, by def. of $\text{map}(\llbracket \tilde{C}' \rrbracket)$, that $\dot{s}'_{n'} = \mathcal{NF}(\text{rm}_l(\dot{s}_n)) \xrightarrow{a_{l'}} \mathcal{NF}(\text{rm}_l(\text{new}))$. Moreover, by using the fact that $\bar{a}_l \in \mathbf{ns}(\dot{s}_m)$ implies that there exists \hat{s} such that $\dot{s}_m \xrightarrow{\bar{a}_l} \hat{s}$, hence, by def. of $\text{map}_{l,C}(\llbracket P \rrbracket)$, $\dot{s}'_m = \overline{\mathcal{NF}_{l,C}(\dot{s}_m)} \xrightarrow{a_{l'}} \overline{\mathcal{NF}_{l,C}(\hat{s})}$, we can build the trace $tr' \widehat{\ } s'_{r'} = \dot{s}'_{n'} \parallel_{\mathcal{A}} \dot{s}'_m \xrightarrow{a_{l'}} \mathcal{NF}(\text{rm}_l(\text{new})) \parallel_{\mathcal{A}} \overline{\mathcal{NF}_{l,C}(\hat{s})}$. For such a trace there cannot be a continuing trace (a trace starting in the last state of the above trace) including a \surd' transition for the following reason. By def. of $\text{map}(\llbracket \tilde{C}' \rrbracket)$, the state $\mathcal{NF}(\text{rm}_l(\text{new}))$ has no outgoing transitions, hence in any continuing trace the test cannot do a \surd transition (that requires synchronization), hence also it cannot do a \surd' transition because, by def. of $\text{map}_{l,C}(\llbracket P \rrbracket)$, \surd' transitions can only be done in the target state $\overline{\mathcal{NF}_{l,C}(\text{newacc})}$ of \surd transitions.
- 2 If $a_{l'} \notin I([C]_l)$, then, since we have also that $\bar{a}_{l'} \in \mathbf{ns}(\dot{s}_m)$, we derive, by def. of $\text{map}_{l,C}(\llbracket P \rrbracket)$, that $\dot{s}'_m = \overline{\mathcal{NF}_{l,C}(\dot{s}_m)} \xrightarrow{\tau} \overline{\mathcal{NF}_{l,C}(\text{new})}$. We build the trace $tr' \widehat{\ } s'_{r'} = \dot{s}'_{n'} \parallel_{\mathcal{A}} \dot{s}'_m \xrightarrow{\tau} \dot{s}'_{n'} \parallel_{\mathcal{A}} \overline{\mathcal{NF}_{l,C}(\text{new})}$. For such a trace there cannot be a continuing trace (a trace starting in the last state of the above trace) including a \surd' transition because, by def. of $\text{map}_{l,C}(\llbracket P \rrbracket)$, the state $\overline{\mathcal{NF}_{l,C}(\text{new})}$ of the test has no outgoing transitions.

We now show that there exists a trace from s_r that leads to a \surd transition. Since we have that $\mathcal{NF}(\tilde{C}') \mathbf{shd} \overline{\mathcal{NF}_{l,C}(P)}$, there exists $tr'' \in Tr_{r''}^{\llbracket \mathcal{NF}(\tilde{C}') \rrbracket \parallel_{\mathcal{A}} \overline{\mathcal{NF}_{l,C}(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 $\dot{t}r'' \in Tr_{n''}^{\llbracket \mathcal{NF}(\tilde{C}') \rrbracket}$ with $\dot{s}''_0 = \dot{s}'_{n'}$ and $\ddot{t}r'' \in Tr_{m''}^{\llbracket \overline{\mathcal{NF}_{l,C}(P)} \rrbracket}$, with $\dot{s}''_0 = \dot{s}'_m$ corresponding to the local moves performed by the two parallel processes when doing trace tr'' . We must have $\dot{\lambda}''_{m''} = \surd'$ and, by def. of $\text{map}_{l,C}(\llbracket P \rrbracket)$, $\dot{s}''_{m''-1} = \overline{\mathcal{NF}_{l,C}(\text{newacc})}$ and, since tr' did not perform a final \surd transition because tr did not perform it, ($r'' \geq 2$ and) $\dot{\lambda}''_{m''-1} = \surd$. Moreover we must have $\text{vis}(\dot{t}r'') = \text{vis}(\text{less}(\ddot{t}r''))$, 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 afterwards) because target states of \surd transition have no outgoing transitions in the semantics of contracts and the transformation $map(\llbracket \tilde{C}' \rrbracket)$ cannot add outgoing τ transitions to such states.

Let us now consider $\dot{tr}''' = [map^{-1}(tr'')]_i \in Tr_{n'''}^{\llbracket \tilde{C}' \rrbracket_i}$. We have $\dot{s}_0'' = \dot{s}'_{n'} = \mathcal{NF}(rm_i(\dot{s}_0'''))$ and $\dot{\lambda}_{n'''}''' = \surd$. Let us also consider $\ddot{tr}''' = map_l^{-1}(less(\dot{tr}''')) \in Tr_{m''-1}^{\llbracket P \rrbracket}$. We have $\ddot{s}_0'' = \dot{s}'_m = \mathcal{NF}_{l,C}(\ddot{s}_0''')$ and $\ddot{\lambda}_{m''-1}''' = \surd$. Moreover, $vis(\dot{tr}''') = vis(less(\dot{tr}'''))\{\bar{a}_i/\bar{a} \mid a \in \mathcal{N}\} = vis(\dot{tr}''')\{a_i/a \mid a \in \mathcal{N}\} = vis(\dot{tr}''')$.

Therefore, there exists $tr''' \in Tr_{r'''}^{\llbracket \tilde{C}' \rrbracket_i \parallel P}$ such that $\lambda_i''' = \tau \forall 1 \leq i \leq r''' - 1$, with $s_0''' = \dot{s}_0''' \parallel_{\mathcal{A}} \ddot{s}_0''' = \dot{s}_n \parallel_{\mathcal{A}} \ddot{s}_m = s_r$ and $\lambda_{r'''}''' = \surd$.