Contracts for Abstract Processes in Service Composition*

Maria Grazia Buscemi
IMT Lucca Institute for Advanced Studies, Italy
m.buscemi@imtlucca.it

Hernán Melgratti
FCEyN, University of Buenos Aires, Argentina
CONICET
hmelgra@dc.uba.ar

Contracts are a well-established approach for describing and analyzing behavioral aspects of web service compositions. The theory of contracts comes equipped with a notion of compatibility between clients and servers that ensures that every possible interaction between compatible clients and servers will complete successfully. It is generally agreed that real applications often require the ability of exposing just partial descriptions of their behaviors, which are usually known as abstract processes. We propose a formal characterization of abstraction as an extension of the usual symbolic bisimulation and we recover the notion of abstraction in the context of contracts.

1 Introduction

Service Oriented Computing is a paradigm that builds upon the notion of services as interoperable elements that can be dynamically discovered through a public description of their interface, which includes their behavior or *contract*. Session types [10, 7, 8] and contracts [11, 4, 5, 2] provide a framework for checking whether a client is compliant with a service and whether a process can be "safely" replaced with another one. Both contracts and session types statically ensure the successful completion of every possible interaction between compatible clients and services.

In a previous work [3] we have addressed an issue related to contracts by developing a formal theory of *abstract processes* in orchestration languages. An *orchestrator* describes the execution flow of a single party in a composite service. The execution of an orchestrator takes control of service invocation, handles service answers and data flow among the different parties in the composition. Since orchestrators are descriptions at implementation level and may contain sensitive information that should be kept private to each party, orchestration comes equipped with the notion of abstract process, which enables the interaction of parties while hiding private information. Essentially, abstract processes are partial descriptions intended to expose the protocols followed by the actual, concrete processes. Typically, abstract processes are used for slicing the interactions of a concrete process over a fixed set of ports. As a sample scenario, consider an organization that sells goods that are produced by another company. The process that handles order requests can be written as follows.

$$C_1 \stackrel{def}{=} order(desc).\overline{askProd}\langle desc\rangle.answProd(cost).\overline{reply}\langle cost \times 1.1\rangle$$

The process C_1 starts by accepting an order as a message on port *order*. Then, the received order is forwarded to the actual producer to obtain a quotation. Finally, the client request is answered by sending the production cost incremented by a 10%. An abstract process of C_1 should at the same time hide the sensitive details of the organization and give enough information to the client for allowing interaction. For

*Research supported by the EU FET-GC2 IST-2004-16004 Integrated Project SENSORIA

Benoît Caillaud and Axel Legay (Eds.):
Foundations of Interface Technologies (FIT'10)
EPTCS 46, 2011, pp. 9–27, doi:10.4204/EPTCS.46.2

© M.G. Buscemi & H. Melgratti
This work is licensed under the
Creative Commons Attribution License.

instance, the following abstract process (where τ stands for a silent, hidden action) shows the interaction of C_1 with a client.

$$A_{C_1} \stackrel{def}{=} order(desc).\tau.\tau.\overline{reply}\langle cost\rangle$$

Another feature of abstract processes is to hide particular values and internal decisions made by concrete processes. Consider, e.g., the following process for authorizing loans.

$$C_2 \stackrel{def}{=} request(amount, salary).\texttt{if } (salary > amount/50) \texttt{ then } \overline{refuse}\langle\rangle \texttt{ else } \overline{approved}\langle\rangle$$

Suppose also that the bank does not want to publicly declare its policy, under which a loan is approved only when the requested amount is at most 50 times the solicitor's salary. This can be achieved by providing an abstract process where some values are *opaque* (noted with $\Box$), i.e., not specified. An abstract process of C_2 can be as below.

$$A_{C_2} \stackrel{def}{=} request(amount, salary).\texttt{if } salary > \Box \texttt{ then } \overline{refuse}\langle\rangle \texttt{ else } \overline{approved}\langle\rangle$$

Note that the conditional process in A_{C_2} has to be thought of as an internal, non-deterministic choice in which the bank may decide either to approve or to refuse the application. In other words, the client cannot infer from A_{C_2} the actual decision that the bank will take. In general, we require an abstraction to provide enough information to decide whether a client and a service are compliant, i.e., whether their interaction will allow them to complete their execution or not.

In [3] we have characterized the valid abstractions of a concrete orchestration and we have shown that valid abstractions preserve compliance. More precisely, we have formally defined suitable abstractions of concrete processes as a relation among abstract and concrete processes, called *simulation-based abstraction* relation, which is an extension of the usual symbolic bisimulation [9, 1].

A main goal of the present paper is to investigate the relation between simulation-based abstraction and contracts. In particular, we aim at recovering the notion of abstraction in the context of the theory of contracts developed in [5]. Contracts are types describing the external, visible behavior of a service. Contracts come equipped with a notion of service compatibility that characterizes all the valid clients of a service, i.e., the clients that terminate any possible interaction with the service. In this sense, contracts can be used to statically ensure that the composition of two services is safe. Contract compatibility induces a preorder relation ($\prec$) among contracts that characterizes the safely replacement of services. For instance, considering two contracts σ_1 and σ_2, if $\sigma_1 \prec \sigma_2$ we know that any valid client of σ_1 is also a valid client of σ_2, hence σ_1 can be substituted by σ_2 in any context. A contract for the service C_1 corresponding to the selling company example introduced above can be written as follows.

$$\sigma_1 \stackrel{def}{=} order.\overline{askProd}.answProd.\overline{reply}$$

Note that σ_1 describes the interactions of C_1 with both the client and the producer. Hence, we would like to use the idea of abstraction in the context of contracts to obtain slices of the behaviour of a service and to reason about the interactions of a service with a particular partner, i.e., we would like to use σ_1 to conclude that any client behaving as $\rho_1 = \overline{order}.reply$ is compliant with the role client of the service σ_1.

More in detail, given a contract σ and a role, defined in terms of a set of visible actions V, the abstraction $\mathscr{A}_V(\sigma)$ of σ can be thought as the contract that hides all the actions in σ that do not appear in V. For instance, the abstraction of σ_1 for the role client will be as follows

$$\mathscr{A}_{\{order,reply\}}(\sigma_1) \stackrel{def}{=} order.\overline{reply}$$

Another key property of the abstraction type is to turn guarded choices into internal choices, if some guards are hidden. For instance, consider the process $P \equiv a().\overline{c}\langle\rangle + b().\overline{d}\langle\rangle$. The type of P is $\sigma = a.\overline{c} + b.\overline{d}$. If we hide a, the abstraction type of σ is

$$\mathscr{A}_{\{b,c,d\}}(\sigma) \stackrel{def}{=} \overline{c} \oplus b.\overline{d}$$

The main technical contributions of this work are the following. Firstly, we define abstraction as a function $\mathscr{A}_V(\_)$ over contracts and show that our definition behaves well with respect to safe replacement, i.e., $\mathscr{A}_V(\sigma)$ can be substituted by $\mathscr{A}_V(\rho)$ whenever σ can be substituted by ρ. Technically speaking, this fact amounts to proving that $\sigma \prec \rho$ implies $\mathscr{A}_V(\sigma) \prec \mathscr{A}_V(\rho)$, when taking $\prec$ as the strong subcontract preorder.

Then, we show that contract abstraction can be used on top of contracts to reason about slices of a concrete service. That is, given a suitable type system for assigning contracts to concrete services, the type of a particular slice of a concrete service can be defined simply as the abstraction of the original contract. Formally, we show that any consistent type system enriched with a typing rule that assigns any slice of a concrete service with the corresponding contract abstraction is a consistent type system. This result allows us to use abstraction over contracts to reason about slices of a concrete service, e.g., we can use $\mathscr{A}_{\{order,reply\}}(\sigma_1)$ to safely reason about the interactions of C_1 with a client.

Finally, we show that contract abstraction matches simulation-based abstraction. Consider the simulation-based abstraction Q of P that characterizes a particular slice of P. Assume that Q has contract σ and consider any compliant client C of Q (namely, the type of C is compliant with σ). Our results ensure that C is also compliant with the slice of P described by Q.

2 Abstract processes

In this section we recall the language of abstract processes proposed in [3] along with a notion of abstraction relation over processes, which is a generalisation of [1]. First, we introduce the language of *abstract processes*, which is a version of value-passing CCS [12] with input guarded choices and conditional statements but without recursion plus the possibility of having *opaque* definitions. An opaque element is meant to hide the precise value of an element: for instance, an opaque assignment to a data variable hides the assigned value. We assume the set of data values to be finite so that the present version of the calculus can be encoded into the fragment without value-passing. We refer the interested reader to [12] for a more detailed treatment.

Syntax We assume an infinite denumerable set of names $\mathscr{N}$ that is partitioned into a set of port names $\mathscr{X}$, a set of finite data variables $\mathscr{V}$, and a finite set of data constants $\mathscr{C}$. We write the special name $\Box$ to denote an opaque element, and we assume $\Box \notin \mathscr{N}$. We let η range over $\mathscr{N} \cup \Box$, $u, v, \ldots$ range over $\mathscr{V}$, $a, b, c, \ldots$ range over $\mathscr{C} \cup \{\Box\}$, and $x, y, z, \ldots$ range over $\mathscr{X}$. We let $m, n, \ldots$ range over $\mathscr{V} \cup \mathscr{C} \cup \{\Box\}$. We write $\tilde{\eta}$ for a tuple of names. *Substitutions*, ranged over by σ, are partial maps from $\mathscr{V}$ onto $\mathscr{V} \cup \mathscr{C} \cup \{\Box\}$. Domain and co-domain of σ, noted $dom(\sigma)$ and $cod(\sigma)$, are defined as usual. By $m\sigma$ we denote $\sigma(m)$ if $m \in dom(\sigma)$, and m otherwise.

The set of *abstract processes* P is given by the following grammar:

$$P \;::=\; 0 \;\big|\; P|P \;\big|\; \tau.P \;\big|\; \overline{x}\langle\widetilde{m}\rangle.P \;\big|\; x_1(\widetilde{v_1}).P + \ldots + x_n(\widetilde{v_n}).P \;\big|\; \texttt{if } m = n \texttt{ then } P \texttt{ else } P$$

As usual, 0 stands for the inert process, $P|P$ for the parallel composition of processes, $\tau.P$ for the process that performs a silent action and then behaves like P, $\overline{x}\langle\widetilde{m}\rangle.P$ for the process that sends the message

m over the port x and then becomes P. The process $x_1(\widetilde{v_1}).P_1 + \ldots + x_n(\widetilde{v_n}).P_n$ denotes an external choice in which some process $x_i(\widetilde{v_i}).P_i$ is chosen when the corresponding guard $x_i(\widetilde{v_i})$ is enabled. The conditional process $\texttt{if } m = n \texttt{ then } P \texttt{ else } P'$ behaves either as P if m and n are syntactically equivalent, or as P' otherwise. Opaque names can appear either as subjects of input and output prefixes, values of output prefixes, or parts of conditions in $\texttt{if \_ then \_ else \_}$ processes, but not as a bound variables. A conditional statement becomes an *internal* choice when at least one value in the condition is opaque; similarly, a guarded choice becomes an internal choice when the subject of the input guard is the opaque name.

We let $P, Q, R \ldots$ range over abstract processes and we simply write process to denote an abstract process. By *concrete* processes we denote processes not containing opaque names. Note that in $x_1(\widetilde{v_1}).P_1 + \ldots + x_n(\widetilde{v_n}).P_n$, the data variables v_i are bound, for all i. We use the standard notions of *free* and *bound* names of processes, noted respectively as $fn(P)$ and $bn(P)$, and α-conversion on bound names. We assume that the sets of free and bound names are disjoint and that the bound names of a process are all distinct from each other. As usual, a process P is *closed* if $fn(P) \cap \mathscr{V} = \emptyset$. We also adopt the usual convention of omitting trailing 0's.

2.1 Symbolic semantics

For the purpose of this paper we only recall the *symbolic* labeled transition relation over processes, while we report in Appendix A the non-symbolic semantics along with a proof that the two semantics are equivalent.

We define *structural congruence*, $\equiv$, as the least congruence over processes that is closed with respect to α-conversion and such that the set of process is a monoid with respect to parallel composition $|$ (being 0 the neutral element).

We let *symbolic actions* λ range over the *silent move*, *input* and *free output* and we let *conditions* M range over a language of Boolean formulas:

$$\lambda ::= \tau \mid x(\widetilde{v}) \mid \overline{x}\langle\widetilde{m}\rangle \qquad M ::= \mathit{true} \mid \mathit{false} \mid m = n \mid m \neq n \mid M \wedge M \mid M \vee M.$$

As usual, for $\lambda \neq \tau$, $subj(\lambda)$ and $obj(\lambda)$ denote the subject and the object of λ respectively. The notions of *free* names $fn(\cdot)$, *bound* names $bn(\cdot)$, and α-conversion over actions and conditions are as expected, considering that the occurrences of the names v_i's are bound in $x(\widetilde{v})$ and that conditions have no bound names. For X a process or an action, $X\sigma$ denotes the expression obtained by replacing in X each data variable $u \in fn(X)$ with $u\sigma$, possibly α-converting to avoid name capturing. By $M\sigma$ we mean the condition obtained by simultaneously replacing in M each data variable $v \in fn(M)$ with $v\sigma$. A condition M is *ground* if M does not contain data variables. The *evaluation* $Ev(M)$ of a ground condition M into the set $\{\mathit{true}, \mathit{false}\}$ is defined by extending in the expected homomorphical way the following clauses:

$$\begin{array}{llll} Ev(\mathit{true}) = \mathit{true} & Ev(a = a) = \mathit{true} & & Ev(a = b) = \mathit{true} \text{ if } \{a,b\} \cap \Box \neq \emptyset \\ Ev(\mathit{false}) = \mathit{false} & Ev(a = b) = \mathit{false} \text{ if } a, b \neq \Box & Ev(a \neq b) = \mathit{true} & \text{if } \{a,b\} \cap \Box \neq \emptyset \end{array}$$

A substitution σ *respects* M, written $\sigma \models M$, if $M\sigma$ is ground and $Ev(M\sigma) = \mathit{true}$. A condition M is *consistent* if there is a substitution σ such that $\sigma \models M$. A condition M *logically entails* a condition N, written $M \Rightarrow N$, if, for every σ, $\sigma \models M$ implies $\sigma \models N$. For instance, $v = a \wedge u \neq b \wedge v = u \Rightarrow a \neq b$ and $\mathit{true} \Rightarrow u = a \vee u \neq a$. For λ a symbolic action and σ a substitution such that every data variable in λ

$$\text{(S-TAU)}\quad \tau.P \xrightarrow{true,\tau} P \qquad \text{(S-OUT)}\quad \overline{x}\langle\widetilde{m}\rangle.P \xrightarrow{true,\overline{x}\langle\widetilde{m}\rangle} P \qquad \text{(S-IN)}\quad x_1(\widetilde{v_1}).P_1+\ldots+x_n(\widetilde{v_n}).P_n \xrightarrow{true,x_i(\widetilde{v_i})} P_i$$

$$\text{(S-PAR)}\quad \frac{P \xrightarrow{M,\lambda} P' \quad bn(\lambda)\cap fn(Q)=\emptyset}{P \mid Q \xrightarrow{M,\lambda} P' \mid Q} \qquad \text{(S-IF)}\quad \frac{P \xrightarrow{M,\lambda} P' \quad m=n\wedge M \text{ consistent}}{\texttt{if } m=n \texttt{ then } P \texttt{ else } Q \xrightarrow{m=n\wedge M,\lambda} P'}$$

$$\text{(S-STR)}\quad \frac{P\equiv Q \quad Q \xrightarrow{M,\lambda} Q' \quad Q'\equiv P'}{P \xrightarrow{M,\lambda} P'} \qquad \text{(S-ELSE)}\quad \frac{Q \xrightarrow{M,\lambda} Q' \quad m\neq n\wedge M \text{ consistent}}{\texttt{if } m=n \texttt{ then } P \texttt{ else } Q \xrightarrow{m\neq n\wedge M,\lambda} Q'}$$

$$\text{(S-CHOICE-1)}\quad \frac{P \xrightarrow{M,\lambda} P' \quad \square\in\{m,n\}}{\texttt{if } m=n \texttt{ then } P \texttt{ else } Q \xrightarrow{M,\lambda} P'} \qquad \text{(S-CHOICE-2)}\quad \frac{Q \xrightarrow{M,\lambda} Q' \quad \square\in\{m,n\}}{\texttt{if } m=n \texttt{ then } P \texttt{ else } Q \xrightarrow{M,\lambda} Q'}$$

$$\text{(S-CHOICE-3)}\quad x_1(\widetilde{v_1}).P_1+\ldots+\square(\widetilde{v_i}).P_i+\ldots+x_n(\widetilde{v_n}).P_n \xrightarrow{true,\tau} P_i$$

Table 1: Symbolic LTS for processes

belongs to $dom(\sigma)$, we write $\lambda\sigma$ to denote the following action:

$$\lambda\sigma \stackrel{\text{def}}{=} \begin{cases} \tau & \text{if } \lambda=\tau \\ \overline{x}\langle a_1,\ldots,a_k\rangle & \text{if } \lambda=\overline{x}\langle n_1,\ldots,n_k\rangle \text{ and } a_i=n_i\sigma \text{ for } i=1,\ldots,k \\ x\langle a_1,\ldots,a_k\rangle & \text{if } \lambda=x(v_1,\ldots,v_k) \text{ and } a_i=\sigma(v_i) \text{ for } i=1,\ldots,k \end{cases}$$

By $\lambda=\lambda'$ we denote the following condition:

$$\lambda=\lambda' \stackrel{\text{def}}{=} \begin{cases} true & \text{if } \lambda=\lambda'=\tau \text{ or } \lambda=\lambda'=x(\widetilde{v}) \\ \widetilde{m}=\widetilde{n} & \text{if } \lambda=\overline{x}\langle\widetilde{m}\rangle \text{ and } \lambda'=\overline{x}\langle\widetilde{n}\rangle \\ false & \text{otherwise} \end{cases}$$

For M a condition and $D=\{M_1,\ldots,M_n\}$ a finite set of conditions, D is a *M-decomposition* if $M \Rightarrow M_1\vee\ldots\vee M_n$. For instance, $\{u=a, u\neq a\}$ is a *true*-decomposition.

The symbolic labeled transition relation $\xrightarrow{M,\lambda}$ over abstract processes is the least relation satisfying the inference rules in Table 1. Intuitively, the condition M in the label M,λ of a transition collects the Boolean constraints on the free data variables of the source process necessary for action λ to take place. For instance, the rules for prefixes say that each prefix can be consumed unconditionally, while rules (S-IF) and (S-ELSE) make the equalities or inequalities of the conditional statements explicit. For instance, the process $P\equiv x(v).\texttt{if } v=a \texttt{ then } \overline{y}\langle v\rangle \texttt{ else } 0$, after a first step, can make a transition under condition that variable v is equal to a:

$$P \xrightarrow{true,x(v)} \texttt{if } v=a \texttt{ then } \overline{y}\langle v\rangle \texttt{ else } 0 \xrightarrow{v=a,\overline{y}\langle v\rangle} 0$$

As another example, consider the process $R\equiv\square(\tau).P+x(v_2).Q$. By rule (S-CHOICE-3), a possible move for R is $R \xrightarrow{true,x(v_2)} Q$, where the input guard is executed. Another possibility is $R \xrightarrow{true,\tau} P$, where R makes an internal choice.

Non-symbolic semantics. The following definition corresponds to the original semantics proposed in [3]. (Details are in Appendix A.)

Definition 1 (Non-symbolic semantics)**.** *Let P, Q and λ be closed terms. $P \xrightarrow{\lambda} Q$ iff $P \xrightarrow{M,\lambda'} P'$, $\sigma \models M$, $\lambda = \lambda'\sigma$ and $Q = P'\sigma$.*

2.2 Simulation-based abstraction

Definition 2 (visible names)**.** *Given a set of visible names V and a symbolic action λ, the set of visible received names of λ, written $vn(\lambda)_V$, is defined as follows:*

$$vn(\lambda)_V \stackrel{\text{def}}{=} \begin{cases} \widetilde{u} & \text{if } \lambda = x(\widetilde{u}) \text{ and } x \in V \\ \emptyset & \text{otherwise} \end{cases}$$

We will omit the subscript V when it is clear from the context.

Definition 3 (simulation-based abstraction)**.** *The family $\mathscr{R} = \{\mathscr{R}^V_M\}_M$ of process relations is a* family of simulation-based abstraction relations*, indexed over the set of conditions M, iff for all M and $P\mathscr{R}^V_M Q$:*

1. *If $Q \xrightarrow{N,\lambda} Q'$ and $bn(\lambda) \cap fn(P,Q,M) = \emptyset$ then there exists a $M \wedge N$-decomposition D s.t. $\forall M' \in D$ there exists $P \xrightarrow{N',\lambda'} P'$, with $M' \Rightarrow N' \wedge \lambda_{|V} = \lambda'$ and $P'\mathscr{R}^{V \cup vn(\lambda)}_{M'} Q'$.*
2. *if $P \xrightarrow{N,\lambda} P'$ and $bn(\lambda) \cap fn(P,Q,M) = \emptyset$ then there exists a $M \wedge N$-decomposition D s.t. $\forall M' \in D$ there exists $Q \xrightarrow{N',\lambda'} Q'$ with $M' \Rightarrow N'_{|V} \wedge \lambda = \lambda'_{|V}$ and $P'\mathscr{R}^{V \cup vn(\lambda')}_{M'} Q'$.*

A process P is a simulation-based abstraction *of a process Q with respect to a set $V \subseteq \mathscr{N}$, written $P \propto^V Q$, if there is an abstraction relation $\mathscr{R}^V_{true}$ s.t. $P\mathscr{R}^V_{true} Q$, with $fn(P) \subseteq V$.*

Condition 1 above states that the abstraction P simulates the concrete process Q up to hidden names. Note that we require $\lambda_{|V} = \lambda'$ instead of the standard definition of symbolic bisimulation that imposes the exact matching of action labels. Condition 2 states that the (concrete) process Q can simulate its abstraction P if we forget about the constraints involving hidden values. That is, if P proposes a move with label $\langle N, \lambda \rangle$ we allow Q to mimic the behavior for a more restrictive condition N'. (Actually, N' may contain several additional constraints involving hidden names.) Note that this makes the abstraction relation not symmetric. For instance, consider the two processes below:

$$P \equiv \texttt{if } v = \Box \texttt{ then } \overline{y}\langle v \rangle \texttt{ else } \overline{z}\langle v \rangle \quad Q \equiv \texttt{if } v = a \texttt{ then } \overline{y}\langle v \rangle \texttt{ else } \overline{z}\langle v \rangle.$$

It holds that $P \propto^V Q$ for $V = \{v, y, z\}$. Indeed, when considering the transition $P \xrightarrow{true,\overline{y}\langle v \rangle} 0$, we can take $Q \xrightarrow{v=a,\overline{y}\langle v \rangle} 0$ since $true \Rightarrow (v = a)_{|V} \wedge \overline{y}\langle v \rangle = \overline{y}\langle v \rangle_{|V}$. Conversely, $P \not\propto^V Q' \equiv \texttt{if } a = a \texttt{ then } \overline{y}\langle v \rangle \texttt{ else } \overline{y}\langle v \rangle$ because $P \xrightarrow{true,\overline{z}\langle v \rangle} 0$ but $Q' \not\xrightarrow{M,\overline{z}\langle v \rangle}$. We remark that the relation $\propto$ is a simulation (since the abstract process simulates the concrete one) but, in general, is not either a bisimulation or a similarity.

3 Theory of contracts

This section summarizes the basics about the theory of contracts proposed in [4, 5]. Let $\mathscr{N}$ be a set of names, the set of contracts Σ is given by the following grammar.

$\alpha ::= a \mid \overline{a} \qquad a \in \mathscr{N}$
$\sigma ::= 0 \mid \alpha.\sigma \mid \sigma \oplus \sigma \mid \sigma + \sigma$

The contract 0 describes a service that does not perform any action. The contract $\alpha.\sigma$ stands for a service that is able to execute α and then continues as σ. The contract $\sigma+\rho$ describes a service that lets the client decide whether to continue as σ or as ρ, while $\sigma \oplus \rho$ stands for a service that internally decides whether to continue as σ or ρ. As usual, trailing 0's are omitted. Contracts will be considered modulo associativity of each sum operator. We usually write summations $\sigma_1+\sigma_2+\ldots+\sigma_n$ and $\sigma_1 \oplus \sigma_2 \oplus \ldots \oplus \sigma_n$ respectively as $\Sigma_{i\in\{1,\ldots,n\}}\sigma_i$ and $\bigoplus i \in \{1,\ldots,n\}\sigma_i$. By convention, $\Sigma_{i\in\emptyset}\sigma_i = 0$.

In this paper we restrict our attention to finite contracts, although the presentation in [5] deals also with infinite contracts in the form of infinite trees that satisfy regularity and a contractivity condition.

The operational semantics of contracts is given in terms of the LTS defined below.

Definition 4 (Transition). *Let $\sigma \overset{\alpha}{\not\mapsto}$ be the least relation such that:*

$$0 \overset{\alpha}{\not\mapsto} \qquad \frac{\alpha \neq \beta}{\beta.\sigma \overset{\alpha}{\not\mapsto}} \qquad \frac{\sigma \overset{\alpha}{\not\mapsto} \quad \rho \overset{\alpha}{\not\mapsto}}{\sigma \oplus \rho \overset{\alpha}{\not\mapsto}} \qquad \frac{\sigma \overset{\alpha}{\not\mapsto} \quad \rho \overset{\alpha}{\not\mapsto}}{\sigma + \rho \overset{\alpha}{\not\mapsto}}$$

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

$$\alpha.\sigma \overset{\alpha}{\mapsto} \sigma \qquad \frac{\sigma \overset{\alpha}{\mapsto} \sigma' \quad \rho \overset{\alpha}{\mapsto} \rho'}{\sigma+\rho \overset{\alpha}{\mapsto} \sigma' \oplus \rho'} \qquad \frac{\sigma \overset{\alpha}{\mapsto} \sigma' \quad \rho \overset{\alpha}{\not\mapsto}}{\sigma+\rho \overset{\alpha}{\mapsto} \sigma'} \qquad \frac{\sigma \overset{\alpha}{\mapsto} \sigma' \quad \rho \overset{\alpha}{\mapsto} \rho'}{\sigma \oplus \rho \overset{\alpha}{\mapsto} \sigma' \oplus \rho'} \qquad \frac{\sigma \overset{\alpha}{\mapsto} \sigma' \quad \rho \overset{\alpha}{\not\mapsto}}{\sigma \oplus \rho \overset{\alpha}{\mapsto} \sigma'}$$

and closed under mirror cases for the external and internal choices.

The operational semantics for contracts handles choices differently from the standard CCS transition system. Traditional CCS rules for a choice commits to the execution of a branch as soon as it performs the first action of the branch, e.g., $a.b+a.c$ reduces to both b and c. Differently, the contract $a.b+a.c$ has only the continuation $b \oplus c$, i.e., the operational semantics does not provide any information about the actual choice that has been taken, in this way the environment is aware of the fact that the system will internally decide whether to behave as b or c. Consequently, for any action α and contract σ there is at most one contract σ' such that $\sigma \overset{\alpha}{\mapsto} \sigma'$. Let $\sigma \overset{\alpha}{\mapsto} \sigma'$, we write $\sigma(\alpha)$ for the unique continuation of σ after a (i.e., $\sigma(\alpha) = \sigma'$). We use $\mathtt{init}(\sigma)$ to denote the set of actions that can be immediately emitted by σ, i.e., $\mathtt{init}(\sigma) = \{\alpha \mid \exists \sigma' s.t. \sigma \overset{\alpha}{\mapsto} \sigma'\}$.

Definition 5 (Ready sets). *Let $\mathscr{P}_f(\mathscr{N} \cup \overline{\mathscr{N}})$ be the set of finite parts of $\mathscr{N} \cup \overline{\mathscr{N}}$, called* ready sets. *Let also $\sigma \Downarrow R$ be the least relation between contracts $\sigma \in \Sigma$ and ready sets R in $\mathscr{P}_f(\mathscr{N} \cup \overline{\mathscr{N}})$ such that*

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

As usual we make $\overline{\overline{a}} = a$. For a given ready set R, $\mathtt{co}(R)$ stands for its complementary ready set, i.e., $\mathtt{co}(R) = \{\overline{\alpha} \mid \alpha \in R\}$.

3.1 Compliance and subcontract relation

Compliance formally states when the behavior of a client complies with the behavior of a service. It is assumed that the behavior of both the client and the service are described by contracts. There is a reserved special action `e` (for "end") that can occur in client contracts and that represents the ability of the client to successfully terminate. Compliance requires that, whenever no further interaction is possible between the client and the service, the client be in a state where this action is available.

Definition 6 (Strong compliance). *$\mathscr{C}$ is a* strong compliance *relation if $(\rho,\sigma) \in \mathscr{C}$ implies that*

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

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

Once the precise notion of compliance between clients and services has been established, the notion of strong subcontract is defined. A contract σ is a strong subcontract of another contract ρ when all clients compliant with σ are also compliant with ρ. This notion is coinductively defined as follows.

Definition 7 (Strong subcontract)**.** $\mathscr{S}$ *is a* strong subcontract relation *if* $(\sigma, \rho) \in \mathscr{S}$ *implies that*

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

We denote with $\sqsubseteq$ *the largest strong subcontract relation.*

It has been shown in [11] that $\sqsubseteq$ is the *must testing preorder* as defined by [6].

3.2 Assigning contracts to ordinary processes

Contracts are intended as types for describing the behavior of concrete implementations. It is assumed that the observable behavior of concrete implementations is described by a labeled transition so that $P \stackrel{\mu}{\to} P'$ describes the evolution of a process P that performs an action μ and then becomes P'. The performed action μ can be either a visible action (e.g., an input a or an output $\overline{a}$) or an internal, invisible action τ that the process P executes autonomously. Then, it is assumed that clients and servers interact by synchronizing over complementary actions, as it is formally stated below.

Definition 8 (Strong process compliance)**.** *Let* $P \| Q \to P' \| Q'$ *be the least relation defined by the rules:*

$$\frac{P \stackrel{\tau}{\to} P'}{P\|Q \to P'\|Q} \qquad \frac{Q \stackrel{\tau}{\to} Q'}{P\|Q \to P\|Q'} \qquad \frac{P \stackrel{\alpha}{\to} P' \quad Q \stackrel{\overline{\alpha}}{\to} Q'}{P\|Q \to P'\|Q'}$$

The reflexive and transitive closure of $\to$ is written $\Rightarrow$; $P\|Q \to$ stands for $P\|Q \to P'\|Q'$ for some P' and Q'. We write $P\|Q \not\to$ if not $P\|Q \to$. A computation of $P\|Q$ is maximal if either it is infinite or there exists $P_n\|Q_n$ such that $P\|Q \Rightarrow P_n\|Q_n \not\to$. The client P is strongly compliant with the service Q, written $P \dashv Q$, if for every configuration $P_i\|Q_i$ of every maximal computation there exists $j \geq i$ such that either $P_j \stackrel{\alpha}{\to} P_{j+1}$ for some α or $P_j \stackrel{\tau}{\not\to}$ and $P_j \stackrel{\mathsf{e}}{\to}$.

It is assumed that a type system is given to check that a process P implements the contract σ. This is expressed by the judgment $\vdash P : \sigma$.

Definition 9. *A type system is* consistent *if, whenever* $\vdash P : \sigma$*, we have*

1. $P \stackrel{\tau}{\to} P'$ *implies* $\vdash P' : \sigma'$ *and* $\sigma \sqsubseteq \sigma'$*;*
2. $P \stackrel{\alpha}{\to} P'$ *implies* $\vdash P' : \sigma'$*,* $\sigma \stackrel{\alpha}{\to}$*, and* $\sigma(\alpha) \sqsubseteq \sigma'$*;*
3. P *diverges implies* $\sigma \Downarrow \emptyset$*;*
4. $P \stackrel{\tau}{\not\to}$ *implies* $\sigma \Downarrow R$ *and* $R \subseteq \{\alpha \mid P \stackrel{\alpha}{\to}\}$.

For consistent type systems, the following Lemma has been proved.

Lemma 1 (Subject reduction)**.** *If* $\vdash P : \rho$ *and* $\vdash Q : \sigma$ *and* $\rho \dashv \sigma$ *and* $P\|Q \to P'\|Q'$*, then* $\vdash P' : \rho'$ *and* $\vdash Q' : \sigma'$ *and* $\rho' \dashv \sigma'$.

It has been shown that consistent type systems are sound with respect to compliance, i.e., two processes are guaranteed to be compliant if their types are compliant, as formally stated by the following result.

Theorem 1. *If* $\vdash P : \rho$ *and* $\vdash Q : \sigma$ *and* $\rho \dashv \sigma$ *then* $P \dashv Q$.

4 Abstraction for contracts

We start by introducing a general definition of the notion of slicing or abstraction of concrete processes. We consider the language of concrete processes enriched with an operator that transforms any action over a hidden channel into an internal action. The abstraction operator is defined as follows

$$\mathscr{A}_V[P]$$

where $V \subseteq \mathscr{N}$ is the set of visible actions.

The process $\mathscr{A}_V[P]$ is a slice of P that behaves as P everytime P performs an action over a visible port, while it performs an internal action when the subject of the action executed by P is a hidden channel. Consequently, we assume that the labeled transition system for processes is extended with the following two rules

$$\frac{P \xrightarrow{\alpha} P' \quad \alpha \in V}{\mathscr{A}_V[P] \xrightarrow{\alpha} \mathscr{A}_V[P']} \qquad \frac{P \xrightarrow{\alpha} P' \quad \alpha \notin V}{\mathscr{A}_V[P] \xrightarrow{\tau} \mathscr{A}_V[P']}$$

In addition, we define the effect of applying abstraction $\mathscr{A}_V$ over a contract σ that hides all actions of σ that are not in V.

Definition 10 (Contract abstraction)**.** *The abstraction $\mathscr{A}_V$ of a contract σ, written $\mathscr{A}_V(\sigma)$, is inductively defined as follows:*

$$\begin{array}{rll}
\mathscr{A}_V(0) = 0 & & \\
\mathscr{A}_V(\alpha.\sigma) = \alpha.\mathscr{A}_V(\sigma) & & \text{if } \alpha \in V \\
\mathscr{A}_V(\alpha.\sigma) = \mathscr{A}_V(\sigma) & & \text{if } \alpha \notin V \\
\mathscr{A}_V(\Sigma_{i\in I}\alpha_i.\sigma_i) = \Sigma_{j\in J}\alpha_j.\mathscr{A}_V(\sigma_j) \oplus \bigoplus_{k\in K}\mathscr{A}_V(\sigma_k) & & \\
\qquad \text{with } J = \{i \in I | \alpha_i \in V\} \text{ and } K = \{i \in I | \alpha_i \notin V\} & & \\
\mathscr{A}_V(\bigoplus_{i\in I}\alpha_i.\sigma_i) = \bigoplus_{i\in I}\mathscr{A}_V(\alpha_i.\sigma_i) & &
\end{array}$$

Previous rules state that applying abstraction to a contract is not just removing the hidden actions. In fact, the abstraction of a contract accounts for the fact that a concrete process may commit a choice when executing a hidden action. The most interesting rule is the one for external choices. Note that the abstraction for $\sigma = \Sigma_{i\in I}\alpha_i.\sigma_i$ corresponds to a contract that internally chooses whether to execute an internal action, i.e., some $\alpha_k \notin V$, or to leave the client to select one of the available visible actions $\alpha_j \in V$.

Example 1. *Consider the following variant of the service that handles loan requests described in the Introduction. In this variant, the service asks a third-party service for a recommendation based on client historical records. The third-party service responds back by sending either a positive or a negative feedback. A contract describing the behavior of the concrete service can be written as follows.*

$$\sigma = request.\overline{askadvice}.(negative.\overline{refused} + positive.\overline{approved})$$

The corresponding contract describing the interaction of the service with the client will be

$$\mathscr{A}_{\{request,refused,approved\}}(\sigma) = request.(\overline{refused} \oplus \overline{approved})$$

This abstraction states clearly that the loan service accepts a client request and then decides internally whether to approve or to refuse it. The internal choice in the abstraction reflects the fact that

a service may commit a choice when it interacts over a hidden channel (e.g., it commits to refuse the request when it receives a negative feedback from the third party).

The following three results state properties for contract abstraction that will be used for proving main results of the paper. The next proposition relates the ready sets of the abstraction $\mathscr{A}_V(\sigma)$ with the ready sets of σ.

Proposition 1. *$\mathscr{A}_V(\sigma) \Downarrow S$ if and only if $\sigma \stackrel{\alpha_1}{\longmapsto} \sigma_1 \ldots \stackrel{\alpha_n}{\longmapsto} \sigma_n$, $\alpha_1, .., \alpha_n \notin V$ and $\sigma_n \Downarrow S'$ with $S' \cap V = S$.*

Proof. $\Rightarrow$) The proof follows by straightforward structural induction on σ. $\Leftarrow$) By induction on the length of the derivation. Base case follows by case analysis on the structure of σ. Induction step follows by case analysis on the structure of σ and inductive hypothesis. □

The following proposition characterizes the continuation $\mathscr{A}_V(\sigma)(\alpha)$ of an abstraction.

Proposition 2. *$\mathscr{A}_V(\sigma) \stackrel{\alpha}{\longmapsto} \rho$ if and only if $\alpha \in V$, $\rho = \bigoplus_{\rho_i \in \mathtt{Alc}(\sigma,\alpha,V)} \mathscr{A}_V(\rho_i)$ with*

$$\mathtt{Alc}(\sigma,\alpha,V) = \{\sigma' \mid \sigma \stackrel{\beta_1}{\longmapsto} \sigma_1 \ldots \stackrel{\beta_n}{\longmapsto} \sigma_n \stackrel{\alpha}{\longmapsto} \sigma' \text{ and } \beta_1, \ldots, \beta_n \notin V\} \neq \emptyset.$$

Proof. $\Rightarrow$) The proof follows by straightforward structural induction on σ. $\Leftarrow$) By induction on the length of the derivation. Base case follows by case analysis on the structure of σ. Induction step follows by case analysis on the structure of σ and inductive hypothesis. □

The result below shows that abstraction preserves continuations under visible actions.

Proposition 3. *Let $\sigma \stackrel{\alpha}{\longmapsto}$ and $\alpha \in V$. Then, $\mathscr{A}_V(\sigma) \stackrel{\alpha}{\longmapsto} \mathscr{A}_V(\sigma(\alpha))$.*

Proof. The proof follows by straightforward structural induction on σ. □

The following proposition ensures that abstraction preserves subcontract relation or, in other words, states that if one contract can be safely replaced by another contract, then any possible slice of the original contract can be safely replaced by the corresponding slice of the new contract.

Proposition 4. *If $\sigma \sqsubseteq \rho$ then $\mathscr{A}_V(\sigma) \sqsubseteq \mathscr{A}_V(\rho)$.*

Proof. The proof follows by showing that $\mathscr{S} = \{(\mathscr{A}_V(\sigma), \mathscr{A}_V(\rho)) | \sigma \sqsubseteq \rho\}$ is a subcontract relation. Due to space limitation we omit details here. (We report proof in Appendix B). □

The following two propositions state properties about the continuations of contract abstractions. These two results are used in the proof of the main result of the following section (Proof details can be found in Appendix B).

Proposition 5. *If $\sigma(\alpha) \sqsubseteq \rho$ and $\alpha \notin V$ then $\mathscr{A}_V(\sigma) \sqsubseteq \mathscr{A}_V(\rho)$.*

Proposition 6. *If $\sigma(\alpha) \sqsubseteq \rho$ and $\alpha \in V$ then $\mathscr{A}_V(\sigma)(\alpha) \sqsubseteq \mathscr{A}_V(\rho)$*

Finally, we show how to extend a consistent type system in order to be able to type processes that use abstraction. This is achieved by extending any consistent type system for concrete processes with the following typing rule

$$(\textsc{TypeAbstraction}) \quad \frac{\vdash P : \sigma}{\vdash \mathscr{A}_V[P] : \mathscr{A}_V(\sigma)}$$

Next result shows that the above rule preserves consistency.

Proposition 7. *A consistent type system enriched with rule* (TYPEABSTRACTION) *results in another consistent type system.*

Proof. Let $\vdash P : \sigma$. As regards consistency condition (1), assume $\mathscr{A}_V[P] \xrightarrow{\tau} \mathscr{A}_V[P']$, then either $P \xrightarrow{\tau} P'$ or $P \xrightarrow{\alpha} P'$ with $\alpha \notin V$. When $P \xrightarrow{\tau} P'$, consistency ensures that $\vdash P : \sigma'$ and $\sigma \sqsubseteq \sigma'$. By Proposition 4, $\mathscr{A}_V(\sigma) \sqsubseteq \mathscr{A}_V(\sigma')$. If $P \xrightarrow{\alpha} P'$ then by consistency $\vdash P : \sigma'$ and $\sigma(\alpha) \sqsubseteq \sigma'$. By Proposition 5, $\mathscr{A}_V(\sigma) \sqsubseteq \mathscr{A}_V(\sigma')$. As regards consistency condition (2), assume that $P \xrightarrow{\alpha} P'$ and $\alpha \notin V$. By consistency, $\vdash P : \sigma'$, $\sigma(\alpha) \sqsubseteq \sigma'$. By Proposition 6, $\mathscr{A}_V(\sigma)(\alpha) \sqsubseteq \mathscr{A}_V(\sigma')$. As regards consistency condition (3), assume that $\mathscr{A}_V[P]$ diverges. Then either P diverges or P has an infinite derivation $P \xrightarrow{\alpha_1} P_1 \ldots \xrightarrow{\alpha_n} P_n \xrightarrow{\alpha_{n+1}} \ldots$ with $\alpha_i = \tau$ or $\alpha_i \notin V$. If P diverges, then $P \Downarrow \emptyset$. Therefore, $\mathscr{A}_V[P] \Downarrow \emptyset$. Otherwise, assume P has an infinite derivation $P \xrightarrow{\alpha_1} P_1 \ldots \xrightarrow{\alpha_n} P_n \xrightarrow{\alpha_{n+1}} \ldots$ with $\alpha_i = \tau$ or $\alpha_i \notin V$. By consistency, this implies that there exists an infinite derivation for the contract $\sigma \xmapsto{\alpha_1} \sigma_1 \ldots \xmapsto{\alpha_n} P_n \xmapsto{\alpha_{n+1}} \ldots$ but this is not possible, since we are considering finite contracts. Finally, as regards consistency condition (4), assume that $\mathscr{A}_V[P] \overset{\tau}{\nmapsto}$. Then, $P \overset{\tau}{\nmapsto}$ and $P \overset{\alpha}{\nmapsto}$ for all $\alpha \notin V$. We derive $\sigma \Downarrow R$ where $R \subseteq \{\alpha \mid P \xrightarrow{\alpha}\}$. Moreover $R \subseteq V$ since $P \overset{\alpha}{\nmapsto}$ for all $\alpha \notin V$. By proposition 1, $\mathscr{A}_V[P] \Downarrow R$. Since, $\alpha \in R$ implies $\alpha \in V$, $P \xrightarrow{\alpha}$ implies $\mathscr{A}_V[P] \xrightarrow{\alpha}$. Hence, $R \subseteq \{\alpha \mid \mathscr{A}_V[P] \xrightarrow{\alpha}\}$. □

5 Contracts for abstract processes

In this section we aim at bridging the theories of processes and contracts presented in the previous sections. We remark that although the language of abstract processes is a kind of value-passing CCS, the remaining of this section will consider just finite domains for values, and hence we implicitly will refer to the usual encoding of value-passing CCS into CCS (i.e., we will refer a channel and a tuple of values just as a single action). Moreover, we say an action is a visible action if its subject is a visible name.

We define a type system that assigns contracts to processes and we prove that the proposed type system is consistent according to Definition 9. We use judgments of the form $\vdash P : \sigma$. We report the typing rules in Table 2 (Rules are analogous to the type system for WS-BPEL proposed in [5]). The main idea behind the type system is that types can contain neither τ's nor parallel composition, and that the type of a guarded choice must be an internal choice if its guards are τ's. In this sense, rule (TAU) is as expected. On the other side, rule (PREF) allows recording in the contract any non-τ prefix. Rules (SUM) and (PAR) are the most interesting and account for assigning to both external choice and parallel composition a contract that is a suitable internal choice. Specifically, the type of a choice is obtained as an internal choice between the branches with τ's as prefixes and an external choice of visible prefixed branches. For instance, consider the process $P \equiv a.P_1 + b.P_2 + \tau.P_3$. It holds that $\vdash P : (a.\sigma_1 + b.\sigma_2) \oplus \sigma_3$ for $\vdash P_1 : \sigma_1$, $\vdash P_2 : \sigma_2$, and $\vdash P_3 : \sigma_3$. Rule (PAR) exploits an idea that reminds the expansion lemma, namely the executions performed by a parallel composition $P|Q$ are the sum of the executions of $P_i \,|\, Q$ and $P \,|\, Q_j$, being P_i and Q_j all the continuations of P and Q, respectively. Note that we do not consider the executions resulting from synchronizations of P and Q over complementary actions, as such synchronizations within the same orchestrator are not allowed. Akin to rule (SUM), the type of a parallel composition is the external choice of the non-τ prefixed alternatives and the internal choice of the branches whose prefixes are τ's. Note that rule (PAR) requires to consider all possible computations $P \xrightarrow{\lambda_i} P_i$ and $Q \xrightarrow{\beta_j} Q_j$ and, consequently, this rule is well-defined when we have a finite number of such computations. We remark that the language for concrete and abstract processes that we are considering ensures us that all processes are finitely branching, hence rule (PAR) is well-defined for our target language.

(NIL) $\vdash 0 : 0$

(TAU) $$\frac{\vdash P : \sigma}{\vdash \tau.P : \sigma}$$

(PREF) $$\frac{\vdash P : \sigma \qquad \lambda \neq \tau}{\vdash \lambda.P : \lambda.\sigma}$$

(SUM) $$\frac{\lambda_i \neq \tau \qquad \vdash P_i : \sigma_i \qquad \vdash Q_j : \rho_j}{\vdash \Sigma_{i\in I}\lambda_i.P_i + \Sigma_{j\in J}\tau.Q_j : \Sigma_{i\in I}\lambda_i.\sigma_i \oplus \bigoplus_{j\in J}\rho_j}$$

(PAR) $$\frac{\vdash P_i|Q : \sigma_i \text{ for all } P \xrightarrow{\lambda_i} P_i \qquad \vdash P|Q_j : \rho_j \text{ for all } Q \xrightarrow{\beta_j} Q_j}{\vdash P|Q : (\Sigma_{\lambda_i\neq\tau}\lambda_i.\sigma_i + \Sigma_{\beta_j\neq\tau}\beta_j.\rho_j) \oplus \bigoplus_{\lambda_i=\tau}\sigma_i \oplus \bigoplus_{\beta_j=\tau}\rho_j} \qquad \text{where:} \begin{cases} P \xrightarrow{\lambda_i} P_i \\ Q \xrightarrow{\beta_j} Q_j \end{cases}$$

(COND1) $$\frac{\vdash P : \sigma \qquad \vdash Q : \rho \qquad \square \in \{m,n\}}{\vdash \texttt{if } m = n \texttt{ then } P \texttt{ else } Q : \sigma \oplus \rho}$$

(COND2) $$\frac{\vdash P : \sigma \qquad \square \notin \{m,n\} \qquad m = n}{\vdash \texttt{if } m = n \texttt{ then } P \texttt{ else } Q : \sigma}$$

(COND3) $$\frac{\vdash Q : \rho \qquad \square \notin \{m,n\} \qquad m \neq n}{\vdash \texttt{if } m = n \texttt{ then } P \texttt{ else } Q : \rho}$$

Table 2: Type System for Contracts

As an example, $\vdash (a+\tau)\,|\,(b+c) : (a.\sigma_{b+c} + b.\sigma_{a+\tau} + c.\sigma_{a+\tau}) \oplus \sigma_{b+c}$. Rules (COND1), (COND2), and (COND3) concern the type of conditional statements. More in detail, rule (COND1) applies if $\square \in \{m,n\}$. In this case, the type of the if-then-else is the internal choice between the type of the two possible alternatives. Conversely, rules (COND2) and (COND3) state that m and n are both visible, then the type assigned is the type of the only possible branch.

Theorem 2. *The type system $\vdash P : \sigma$ shown in Table 2 is consistent.*

Proof. The proof is by induction on the structure of P. See appendix B. □

Next result states an auxiliary property that will be used when proving the main result of this section. It states that the reductions of an abstraction of a concrete process are in one-to-one correspondence with the visible reductions of the concrete process.

Proposition 8. *Let P and Q be two closed processes such that $P \propto^V Q$.*

1. *$\mathscr{A}_V[Q] \xrightarrow{\alpha} \mathscr{A}_V[Q']$ implies $P \xrightarrow{\alpha} P'$ and $P' \propto^V Q'$.*
2. *$P \xrightarrow{\alpha} P'$ implies $\mathscr{A}_V[Q] \xrightarrow{\alpha} \mathscr{A}_V[Q']$ and $P' \propto^V Q'$.*

Proof. See Appendix B. □

The following result formalizes the relation among abstractions and strong compliance. It basically states that whenever a client P has a type that is compliant with the type of an abstract process Q which is an abstraction of a concrete process R, then P correctly interacts with the filtered process $\mathscr{A}_V[R]$

Theorem 3. *Let $P:\sigma$, $Q:\rho$ and $Q \propto^V R$. If $\sigma \dashv \rho$ then $P \dashv \mathscr{A}_V[R]$.*

Proof. The proof follows the line of the proof of Theorem 4.5 in [5]. Akin to [5], we reserve a special action e (for "end") that can occur in client contracts and that represents the ability of the client to successfully terminate. Then we require that, whenever no further interaction is possible between the client and the service, the client be in a state where this action is available.

First, we notice that, by Proposition 8, any computation $P||\mathscr{A}_V[R] \to P'||\mathscr{A}_V[R']$ has a corresponding computation $P||Q \to P'||Q'$ with $Q' \propto^V R'$. Because of Lemma 1, we only need to consider maximal computations, i.e., cases in which $P\|\mathscr{A}_V[R] \not\to$ or $P\|\mathscr{A}_V[R]$ diverges (equivalently, cases in which $P\|Q \not\to$ or $P\|Q$ diverges for $Q \propto^V R$). Let $P\|Q \not\to$ and assume, by contradiction, that $P \stackrel{\mathsf{e}}{\not\to}$. From $\sigma \dashv \rho$ we know that $\rho \downarrow R$ implies $R \neq \emptyset$ (by Definition 6). From $P\|Q \not\to$, we have that whenever $P \stackrel{\alpha}{\to}$ we have $Q \stackrel{\alpha}{\not\to}$ and hence $\mathscr{A}_V[R] \stackrel{\alpha}{\not\to}$. Consequently, $\{\alpha | P \stackrel{\alpha}{\to}\} \cap \mathtt{co}(\{\alpha | Q \stackrel{\alpha}{\to}\}) = \emptyset$. From consistency condition (4) there exist R and S such that $\rho \Downarrow R$ and $\sigma \Downarrow S$ and $\mathtt{co}(R) \cap S = \emptyset$ and $\mathsf{e} \notin R$, but this is absurd from the hypothesis that $\rho \dashv \sigma$. Hence $P \stackrel{\mathsf{e}}{\to}$. Assume $P\|Q$ diverges. First, note that P cannot diverge since consistency condition (3) requires that $\rho \Downarrow \emptyset$. Then, the only possibility is $P \not\to$ and Q diverges. By consistency condition (3) we derive $\sigma \Downarrow \emptyset$, hence $\rho \Downarrow R$ implies $\mathsf{e} \in R$. From consistency condition (4) we conclude $P \stackrel{\mathsf{e}}{\to}$. □

6 Conclusions

In this paper we have investigated the relation among the theory of contracts and the hiding of selected actions. We have shown that we can recover the notion of abstraction as a kind of filter over processes and we accommodate this notion into the theory of contracts for web services when considering finite contracts. We remark that the current definition for abstraction is not suitable for handling infinite contracts. In fact, it turns out that abstraction may not preserve the contractivity condition of contracts. In order to see this, consider the contract $\sigma = b + a.b + a.a.b + a.a.a.b + \ldots + a.a.a\ldots$ that accounts for an infinite execution of a's. Contract σ can be written with the recursive expression $rec\ x = a.x + b$. Then, by taking the current definition of abstraction, $\mathscr{A}_{\{b\}}(\sigma)$ will be associated with the recursive equation $rec\ x = x + b$, for which contractivity does not hold. We left as future work the definition of abstraction for infinite contracts.

Acknowledgements The authors thank anonymous reviewers for their helpful comments on an earlier version of this paper.

References

[1] M. Boreale & R. De Nicola (1996): *A symbolic semantics for the Pi-calculus*. *Inform. and Comput.* 126(1), pp. 34–52.

[2] M. Bravetti & G. Zavattaro (2007): *Towards a Unifying Theory for Choreography Conformance and Contract Compliance*. In: *Software Composition*, *Lect. Notes in Comput. Sci.* 4829, Springer Verlag, pp. 34–50.

[3] M.G. Buscemi & H. Melgratti (2009): *Abstract Processes in Orchestration Languages*. In: *ESOP*, *Lect. Notes in Comput. Sci.* 5502, Springer Verlag, pp. 301–315.

$$\text{(TAU)}\quad \tau.P \overset{\tau}{\rightarrowtail} P \qquad \text{(OUT)}\quad \overline{x}\langle\widetilde{a}\rangle.P \overset{\overline{x}\langle\widetilde{a}\rangle}{\rightarrowtail} P \qquad \text{(IN)}\quad x_1(\widetilde{v_1}).P_1+\ldots+x_n(\widetilde{v_n}).P_n \overset{x_i\langle\widetilde{a}\rangle}{\rightarrowtail} P_i\{\widetilde{a}/\widetilde{v_i}\}$$

$$\text{(IF)}\quad \frac{P \overset{\alpha}{\rightarrowtail} P'}{\texttt{if } a=a \texttt{ then } P \texttt{ else } Q \overset{\alpha}{\rightarrowtail} P'} \qquad \text{(ELSE)}\quad \frac{Q \overset{\alpha}{\rightarrowtail} Q' \quad a\neq b}{\texttt{if } a=b \texttt{ then } P \texttt{ else } Q \overset{\alpha}{\rightarrowtail} Q'}$$

$$\text{(PAR)}\quad \frac{P \overset{\alpha}{\rightarrowtail} P'}{P \mid Q \overset{\alpha}{\rightarrowtail} P' \mid Q} \qquad \text{(STR)}\quad \frac{P\equiv Q \quad Q \overset{\alpha}{\rightarrowtail} Q' \quad Q'\equiv P'}{P \overset{\alpha}{\rightarrowtail} P'}$$

$$\text{(CHOICE-1)}\quad \frac{P \overset{\alpha}{\rightarrowtail} P' \quad \square\in\{m,n\}}{\texttt{if } m=n \texttt{ then } P \texttt{ else } Q \overset{\alpha}{\rightarrowtail} P'} \qquad \text{(CHOICE-2)}\quad \frac{Q \overset{\alpha}{\rightarrowtail} Q' \quad \square\in\{m,n\}}{\texttt{if } m=n \texttt{ then } P \texttt{ else } Q \overset{\alpha}{\rightarrowtail} Q'}$$

$$\text{(CHOICE-3)}\quad x_1(\widetilde{v_1}).P_1+\ldots+\square(\widetilde{v_i}).P_i+\ldots+x_n(\widetilde{v_n}).P_n \overset{\tau}{\rightarrowtail} P_i$$

Figure 1: LTS for processes

[4] Giuseppe Castagna, Nils Gesbert & Luca Padovani (2008): *A theory of contracts for web services*. In: *POPL*, pp. 261–272.

[5] Giuseppe Castagna, Nils Gesbert & Luca Padovani (2009): *A theory of contracts for Web services*. *ACM Trans. Program. Lang. Syst.* 31(5). Available at `http://doi.acm.org/10.1145/1538917.1538920`.

[6] R. De Nicola & M. Hennessy (1984): *Testing Equivalences for Processes*. *Theoret. Comput. Sci.* 34, pp. 83–133.

[7] Mariangiola Dezani-Ciancaglini, Dimitris Mostrous, Nobuko Yoshida & Sophia Drossopoulou (2006): *Session Types for Object-Oriented Languages*. In: *Proc. ECOOP*, *Lecture Notes in Computer Science* 4067, Springer, pp. 328–352.

[8] Simon J. Gay & Malcolm Hole (2005): *Subtyping for session types in the pi calculus*. *Acta Informaticae* 42(2–3), pp. 191–225.

[9] M. Hennessy & H. Lin (1995): *Symbolic bisimulations*. *Theoret. Comput. Sci.* 138, pp. 353–389.

[10] Kohei Honda (1993): *Types for Dynamic Interaction*. In: *Proc. CONCUR*, *Lecture Notes in Computer Science* 715, Springer, pp. 509–523.

[11] Cosimo Laneve & Luca Padovani (2007): *The Must Preorder Revisited*. In: *CONCUR*, *Lect. Notes in Comput. Sci.* 4703, pp. 212–225.

[12] R. Milner (1989): *Communication and Concurrency*. Prentice Hall International.

A The non-symbolic semantics of orchestrators

The original definition of the operational semantics of orchestrators as defined in [3] is shown in Figure 1

Following result states the correspondence between the original semantics non-symbolic semantics and the one introduced in Definition 1.

Theorem 4. *Let P be a closed process. $P \overset{\alpha}{\rightarrow} P'$ iff $P \overset{\alpha}{\rightarrowtail} P'$.*

Proof. $\Rightarrow$) By Definition 1, $P \overset{\alpha}{\rightarrow} P'$ implies $P \overset{M,\lambda}{\longrightarrow} Q$ and $\alpha=\lambda\sigma$ and $P'=Q\sigma$ with $\sigma\models M$. The proof follows by straightforward rule induction on the derivaion of $P \overset{M,\lambda}{\longrightarrow} Q$.

– (S-TAU): $P=\tau.Q$, $M=true$, $\lambda=\tau$. For any substitution σ, we have that $\sigma\models M$, $\alpha=\lambda\sigma=true$. Also, P closed implies Q closed, hence $P'=Q\sigma=Q$. By rule (TAU), $P=\tau.Q \overset{\tau}{\rightarrowtail} Q$.
– (S-OUT) **and** (S-CHOICE-3): these cases follow as for (S-TAU).
– (S-IN): $P=x_1(\widetilde{v_1}).P_1+\ldots+x_n(\widetilde{v_n}).P_n$, $M=true$, $\lambda=x_i(\widetilde{v_i})$, $Q=P_i$. Since P is closed, $fn(Q)\subseteq\widetilde{v_i}$. Then, for any $\sigma\models M$, $P'=Q\sigma=P_i\sigma=P_i\sigma_{|\widetilde{v_i}}$ and $\alpha=\lambda\sigma=\lambda\sigma_{|\widetilde{v_i}}$. By rule (IN) $P \overset{x_i(\widetilde{v_i})\sigma_{|\widetilde{v_i}}}{\rightarrowtail} P_i\sigma_{|\widetilde{v_i}}=P'$
– (S-PAR): $P=P_1|P_2$, $P_1 \overset{M,\lambda}{\longrightarrow} P_1'$, $Q=P_1'|P_2$. Since, $\sigma\models M$, by Definition 1 $P_1 \overset{\alpha}{\rightarrow} P_1'\sigma$. By inductive hypothesis, $P_1 \overset{\alpha}{\rightarrowtail} P_1'\sigma$. By rule (PAR) $P_1|P_2 \overset{\alpha}{\rightarrowtail} P_1'\sigma|P_2$. Since P is closed, also P_2 is closed. Hence, $P_2\sigma=P_2$. Therefore, $P_1|P_2 \overset{\alpha}{\rightarrowtail} Q\sigma$
– (S-STR), (S-CHOICE-1) **and** (S-CHOICE-2): these cases follow analogously to (S-PAR).
– (S-IF): Since P is closed, the only possibility for m and n is to be the same constant. Hence, $P=\texttt{if } a=a \texttt{ then } P_1 \texttt{ else } Q_2$, $P_1 \overset{M,\lambda}{\longrightarrow} Q$. By inductive hypothesis, $P_1 \overset{\alpha}{\rightarrowtail} Q\sigma$. Then, by rule (IF), $P \overset{\alpha}{\rightarrowtail} Q\sigma$.
– (S-ELSE): This case is analogous to (S-IF).

⇐) – (TAU): $P=\tau.P'$, $\alpha=\tau$. By (TAU), $P \overset{true,\tau}{\longrightarrow} P'$. Since, P' is closed, $P'\sigma=P'$ for any σ. $P \overset{\tau}{\rightarrow} P'$.
– (OUT): This case follows as (TAU).
– (IN): $P=x_1(\widetilde{v_1}).P_1+\ldots+x_n(\widetilde{v_n}).P_n$, $\alpha=\overline{x_i}\langle\widetilde{a}\rangle$ and $P'=P_i\{\widetilde{a}/\widetilde{v_i}\}$.By rule (IN), $P \overset{true,x_1(\widetilde{v_1})}{\longrightarrow} P_i$. Note that $\{\widetilde{a}/\widetilde{v_i}\}\models true$. Then, by Definition 1, $P \overset{\alpha}{\rightarrow} P_i\{\widetilde{a}/\widetilde{v_i}\}$.
– (IF): $P=\texttt{if } a=a \texttt{ then } P_1 \texttt{ else } P_2$, $P_1 \overset{\alpha}{\rightarrowtail} P'$. By inductive hypothesis, $P_1 \overset{\alpha}{\rightarrow} P'$. By definition, there exist M, Q, σ and λ s.t. $\sigma\models M$, $\alpha=\lambda\sigma$, $P'=Q\sigma$, $P_1 \overset{M,\lambda}{\longrightarrow} Q$. Since M is consistent, $M\wedge a=a$ is consistent. Then, by rule (S-IN), $P \overset{M\wedge a=a,\lambda}{\longrightarrow} Q$. From $\sigma\models M$, we have $\sigma\models M\wedge a=a$. By Definition 1, $P \overset{\alpha}{\rightarrow} P'$.
– (ELSE):Analogous to (IF).
– (PAR): $P=P_1|P_2$ with $P_1 \overset{\alpha}{\rightarrowtail} P_1'$ and $P'=P_1'|P_2$. By inductive hypothesis, $P_1 \overset{\alpha}{\rightarrow} P_1'$. By definition 1, $P_1 \overset{M,\lambda}{\longrightarrow} Q$ and there exists $\sigma\models M$ s.t. $\alpha=\lambda\sigma$ and $P_1=Q\sigma$. By rule (S-PAR), $P_1|P_2 \overset{M,\lambda}{\longrightarrow} Q|P_2$ (side condition holds because P_2 is closed). By definition 1, we have $P_1|P_2 \overset{alpha}{\longrightarrow} (Q|P_2)\sigma$. Since P_2 is closed. $P_2\sigma=P_2$ and, hence, $(Q|P_2)\sigma=P'$
– (STR),(CHOICE-1),(CHOICE-2) **and** (CHOICE-3): Follows by using inductive hypothesis.

□

Proposition 9. *If* $P \overset{M,\lambda}{\longrightarrow} Q$ *then*

- $fn(M)\subseteq fn(P)$.
- $fn(Q)\subseteq fn(P)\cup bn(\lambda)$.
- M *is consistent.*

Proof. It follows by straightforward rule induction. □

Proposition 10. *If* $P\propto_M^V Q$ *and* $fn(P,Q)\cap fn(M)=\emptyset$ *then* $P\propto^V Q$.

Proof. We first fix the following notation: given a constraint M and a set of names S, we write $M\backslash S$ from M by removing all terms containing a name in S. If $P \propto_M^V Q$ then there exists a family of abstraction relations $\{\mathscr{R}_N^V\}_N$ such that $P\mathscr{R}_M^V Q$. We take the following family of relations $\{\mathscr{S}_L^V\}_L$, where $\mathscr{S}_L^V = \mathscr{R}_N^V$ with $L = N\backslash fn(M)$. We now show that this is a family of abstractions relations.

Let P and Q such that $P\mathscr{S}_L^V Q$:

1. Assume $Q \xrightarrow{N_1,\lambda} Q'$ and $bn(\lambda) \cap fn(P,Q,L) = \emptyset$. Without loss of generality we can assume that $bn(\lambda) \cap fn(P,Q,L \cup M) = \emptyset$ (This can always be achieved by alpha-renaming bound names.). We know that $P\mathscr{R}_N^V Q$ with $L = N\backslash fn(M)$. Since $\mathscr{R}_N^V$ is an abstraction relation, there exists a $N \wedge N_1$-decomposition D s.t. $\forall M_1 \in D$ there exists $P \xrightarrow{N_1',\lambda'} P'$, with $M_1 \Rightarrow N_1'$, $\lambda_{|V} = \lambda'$ and $P'\mathscr{R}_{M_1}^{V \cup vn(\lambda)} Q'$. By Proposition 9, $P \xrightarrow{N_1',\lambda'} P'$ implies $fn(N_1') \subseteq fn(P)$. Since, $fn(P) \cap fn(M) = \emptyset$ then $fn(N_1') \cap fn(M) = \emptyset$ for all N_1'. Consequently, $M_1\backslash fn(M) \Rightarrow N_1'$ and $\bigvee_i M_i\backslash fn(M)$ is a L-decomposition.

2. if $P \xrightarrow{N,\lambda} P'$, the proof follows as in the previous case.

□

Proposition 11. *If $P\{a/x\} \xrightarrow{M,\lambda} Q$ then there exist N, λ' and Q' s.t. $M = N\{a/x\}$, $\lambda = \lambda'\{a/x\}$ and $Q = Q'\{a/x\}$.*

Proof. The proof follows by straightforward rule induction. □

Proposition 12. *Let $P \propto_M^{V\cup\{x\}} Q$. For all a s.t. $\{a/x\} \models M$, $P\{a/x\} \propto_M^V Q\{a/x\}$.*

Proof. We take the following family of relations $\{\mathscr{S}_L^V\}_L$, where

$$\mathscr{S}_M^V = \{(P\{a/x\}, Q\{a/x\}) | P\mathscr{R}_M^{V\cup\{x\}} Q\}$$

We show that this is a family of abstractions relations. Assume that $P\{a/x\}\mathscr{S}_M^V Q\{a/x\}$:

1. Assume $Q\{a/x\} \xrightarrow{N_1,\lambda} Q'$ and $bn(\lambda) \cap fn(P,Q,N) = \emptyset$. By Proposition 11, $Q \xrightarrow{N_0,\lambda_0} Q_0$ and $N_1 = N_0\{a/x\}$, $\lambda = \lambda_0\{a/x\}$ and $Q' = Q_0\{a/x\}$. Since $P\mathscr{R}_M^{V\cup\{x\}} Q$, there is a $M \wedge N_0$-decomposition D s.t. $\forall M' \in D$ there exists $P \xrightarrow{N_0',\lambda_0'} P_0'$ with $M' \Rightarrow N_0'$, $\lambda_{|V} = \lambda'$ and $P_0'\mathscr{R}_{M'}^{V\cup\{x\}\cup vn(\lambda)} Q_0$. By definition, $P_0\{a/x\}\mathscr{S}_{M'}^{V\cup vn(\lambda)\backslash\{x\}} Q\{a/x\}$, From $M' \Rightarrow N_0$ we have that $M'\{a/x\} \Rightarrow N_0\{a/x\}$. Since D is a $M \wedge N_0$-decomposition we have $M \wedge N_0 \Rightarrow D$ and hence $(M \wedge N_0)\{a/x\} \Rightarrow D\{a/x\}$. From $\{a/x\} \models M$ we have that $M \wedge N_0\{a/x\} \Rightarrow D\{a/x\}$. Consequently, $D\{a/x\}$ is the requested $M \wedge N_1$-decomposition.

2. if $P\{a/x\} \xrightarrow{N,\lambda} P'\{a/x\}$, the proof follows as in the previous case.

□

B Proofs of the results in Sections 4 and 5

Proof of Proposition 4. The proof follows by showing that $\mathscr{S} = \{(\mathscr{A}_V(\sigma), \mathscr{A}_V(\rho)) | \sigma \sqsubseteq \rho\}$ is a subcontract relation.

1. Assume $\mathscr{A}_V(\rho) \Downarrow R$. By Proposition 1, we have that $\rho \stackrel{\alpha_1}{\longmapsto} \rho_1 \ldots \stackrel{\alpha_n}{\longmapsto} \rho_n$ and $\rho_n \Downarrow R'$ with $R' \cap V = R$. Since $\sigma \sqsubseteq \rho$, there exists $\sigma \stackrel{\alpha_1}{\longmapsto} \sigma_1 \ldots \stackrel{\alpha_n}{\longmapsto} \sigma_n$ with $\sigma_i \sqsubseteq \rho_i$ for $i = 1..n$. hence, there exists $S' \subseteq R'$ such that $\sigma_n \Downarrow S'$. By Proposition 1, $\mathscr{A}_V(\sigma) \Downarrow S$ with $S = S' \cap V$. Since $S' \subseteq R'$ we have that $S = S' \cap V \subseteq R' \cap V = R$.

2. Assume $\rho \stackrel{\alpha}{\longmapsto} \rho'$. By Proposition 2, $\alpha \in V$ and $\rho' = \bigoplus_{\rho_i \in \mathtt{Alc}(\rho,\alpha,V)} \mathscr{A}_V(\rho_i)$ with

$$\mathtt{Alc}(\rho,\alpha,V) = \{\rho' \mid \rho \stackrel{\beta_1}{\longmapsto} \rho_1 \ldots \stackrel{\beta_n}{\longmapsto} \rho_n \stackrel{\alpha}{\longmapsto} \rho' \text{ and } \beta_1,\ldots,\beta_n \notin V\}.$$

   Since $\sigma \sqsubseteq \rho$, for any ρ_i in $\mathtt{Alc}(\rho,\alpha,V)$ there exists a σ_i s.t. $\sigma_i \in \mathtt{Alc}(\sigma,\alpha,V)$, i.e., $\sigma \stackrel{\beta_1}{\longmapsto} \tau_1 \ldots \stackrel{\beta_n}{\longmapsto} \tau_n \stackrel{\alpha}{\longmapsto} \sigma_i$. By proposition 2, $\sigma' = \bigoplus_{\sigma_i \in \mathtt{Alc}(\sigma,\alpha,V)} \mathscr{A}_V(\sigma_i)$. It remains to show that $(\sigma',\rho') \in \mathscr{S}$. This is done by noting that $\sigma' = \bigoplus_j \mathscr{A}_V(\tau_j) \oplus \tau$ such that any $\tau_j \in \mathtt{Alc}(\sigma,\alpha,V)$ and there is a corresponding ρ_j in $\mathtt{Alc}(\rho,\alpha,V)$ and $\tau_j \sqsubseteq \rho_j$. Note that it can be easily proved that $\sigma_1 \sqsubseteq \rho_1$ and $\sigma_2 \sqsubseteq \rho_2$ implies $\sigma_1 \oplus \sigma_2 \sqsubseteq \rho_1 \oplus \rho_2$. Consequently, $\bigoplus_j \tau_j \sqsubseteq \bigoplus_j \rho_j$. We can easily also prove that $\sigma_1 \oplus \sigma_2 \sqsubseteq \sigma_1$ for all σ_1,σ_2. Hence, $\bigoplus_j \tau_j \oplus \tau \sqsubseteq \bigoplus_j \rho_j$, and finally, $(\sigma',\rho') \in \mathscr{S}$ by definition of $\mathscr{S}$

Proof of Proposition 5. The proof follows by showing that $\mathscr{S} = \{(\mathscr{A}_V(\sigma), \mathscr{A}_V(\rho)) | \sigma(\gamma) \sqsubseteq \rho \text{ and } \alpha \in V\}$ is a subcontract relation.

1. Assume $\mathscr{A}_V(\rho) \Downarrow R$. By Proposition 1, we have that $\rho \stackrel{\alpha_1}{\longmapsto} \rho_1 \ldots \stackrel{\alpha_n}{\longmapsto} \rho_n$ and $\rho_n \Downarrow R'$ with $R' \cap V = R$. Since $\sigma(\gamma) \sqsubseteq \rho$, there exists $\sigma(\gamma) \stackrel{\alpha_1}{\longmapsto} \sigma_1 \ldots \stackrel{\alpha_n}{\longmapsto} \sigma_n$ with $\sigma_i \sqsubseteq \rho_i$ for $i = 1..n$. Consequently, $\sigma \stackrel{\gamma}{\longrightarrow} \sigma(\alpha) \stackrel{\alpha_1}{\longmapsto} \sigma_1 \ldots \stackrel{\alpha_n}{\longmapsto} \sigma_n$ with $\sigma_i \sqsubseteq \rho_i$. Hence, there exist $S' \subseteq R'$ such that $\sigma_n \Downarrow S'$. By proposition 1, $\mathscr{A}_V(\sigma) \Downarrow S$ with $S = S' \cap V$. Since $S' \subseteq R'$ we have that $S = S' \cap V \subseteq R' \cap V = R$.

2. Assume $\rho \stackrel{\alpha}{\longmapsto} \rho'$. By Proposition 2, $\alpha \in V$ and $\rho' = \bigoplus_{\rho_i \in \mathtt{Alc}(\rho,\alpha,V)} \mathscr{A}_V(\rho_i)$ with

$$\mathtt{Alc}(\rho,\alpha,V) = \{\rho' \mid \rho \stackrel{\beta_1}{\longmapsto} \rho_1 \ldots \stackrel{\beta_n}{\longmapsto} \rho_n \stackrel{\alpha}{\longmapsto} \rho' \text{ and } \beta_1,\ldots,\beta_n \notin V\}.$$

   Since $\sigma(\gamma) \sqsubseteq \rho$, for any ρ_i in $\mathtt{Alc}(\rho,\alpha,V)$ there exists a σ_i s.t. $\sigma_i \in \mathtt{Alc}(\sigma(\alpha),\alpha,V)$, i.e., $\sigma(\gamma) \stackrel{\beta_1}{\longmapsto} \tau_1 \ldots \stackrel{\beta_n}{\longmapsto} \tau_n \stackrel{\alpha}{\longmapsto} \sigma_i$. Note that $\sigma_i \in \mathtt{Alc}(\sigma(\gamma),\alpha,V)$ implies $\sigma_i \in \mathtt{Alc}(\sigma,\alpha,V)$. By proposition 2, $\sigma' = \bigoplus_{\sigma_i \in \mathtt{Alc}(\sigma,\alpha,V)} \mathscr{A}_V(\sigma_i)$. It remains to show that $(\sigma',\rho') \in \mathscr{S}$. This is done by noting that $\sigma' = \bigoplus_j \mathscr{A}_V(\tau_j) \oplus \tau$ such that any $\tau_j \in \mathtt{Alc}(\sigma,\alpha,V)$ and there is a corresponding ρ_j in $\mathtt{Alc}(\rho,\alpha,V)$ and $\tau_j \sqsubseteq \rho_j$. Note that it can be easily proved that $\sigma_1 \sqsubseteq \rho_1$ and $\sigma_2 \sqsubseteq \rho_2$ implies $\sigma_1 \oplus \sigma_2 \sqsubseteq \rho_1 \oplus \rho_2$. Consequently, $\bigoplus_j \tau_j \sqsubseteq \bigoplus_j \rho_j$. We can easily also prove that $\sigma_1 \oplus \sigma_2 \sqsubseteq \sigma_1$ for all σ_1,σ_2. Hence, $\bigoplus_j \tau_j \oplus \tau \sqsubseteq \bigoplus_j \rho_j$, and finally, $(\sigma',\rho') \in \mathscr{S}$ by definition of $\mathscr{S}$

Proof of Proposition 6. The proof follows by showing that $\mathscr{S} = \{(\mathscr{A}_V(\sigma)(\gamma), \mathscr{A}_V(\rho)) | \sigma(\gamma) \sqsubseteq \rho \text{ and } \gamma \in V\}$ is a subcontract relation.

1. Assume $\mathscr{A}_V(\rho) \Downarrow R$. By Proposition 1, we have that $\rho \stackrel{\alpha_1}{\longmapsto} \rho_1 \ldots \stackrel{\alpha_n}{\longmapsto} \rho_n$ and $\rho_n \Downarrow R'$ with $R' \cap V = R$. Since $\sigma(\gamma) \sqsubseteq \rho$, there exists $\sigma(\gamma) \stackrel{\alpha_1}{\longmapsto} \sigma_1 \ldots \stackrel{\alpha_n}{\longmapsto} \sigma_n$ with $\sigma_i \sqsubseteq \rho_i$ for $i = 1..n$. Hence, there exist $S' \subseteq R'$ such that $\sigma_n \Downarrow S'$. By proposition 1, $\mathscr{A}_V(\sigma(\gamma)) \Downarrow S$ with $S = S' \cap V$. By Proposition 3, $\mathscr{A}_V(\sigma)(\gamma) = \mathscr{A}_V(\sigma(\gamma))$, hence $\mathscr{A}_V(\sigma)(\gamma) \Downarrow S$ with $S = S' \cap V$. Since $S' \subseteq R'$ we have that $S = S' \cap V \subseteq R' \cap V = R$.

2. Assume $\rho \overset{\alpha}{\mapsto} \rho'$. By Proposition 2, $\alpha \in V$ and $\rho' = \bigoplus_{\rho_i \in \mathtt{Alc}(\rho,\alpha,V)} \mathscr{A}_V(\rho_i)$ with

$$\mathtt{Alc}(\rho,\alpha,V) = \{\rho' \mid \rho \overset{\beta_1}{\longmapsto} \rho_1 \ldots \overset{\beta_n}{\longmapsto} \rho_n \overset{\alpha}{\mapsto} \rho' \text{ and } \beta_1,\ldots,\beta_n \notin V\}.$$

Since $\sigma(\gamma) \sqsubseteq \rho$, for any ρ_i in $\mathtt{Alc}(\rho,\alpha,V)$ there exists a σ_i s.t. $\sigma_i \in \mathtt{Alc}(\sigma(\gamma),\alpha,V)$, i.e., $\sigma(\gamma) \overset{\beta_1}{\longmapsto} \tau_1 \ldots \overset{\beta_n}{\longmapsto} \tau_n \overset{\alpha}{\mapsto} \sigma_i$. By proposition 2, $\sigma' = \bigoplus_{\sigma_i \in \mathtt{Alc}(\sigma(\gamma),\alpha,V)} \mathscr{A}_V(\sigma_i)$. Moreover, $\mathscr{A}_V(\sigma) \overset{\gamma}{\to} \sigma'$, i.e. $\mathscr{A}_V(\sigma)(\gamma) = \sigma'$, by Proposition 3. It remains to show that $(\sigma',\rho') \in \mathscr{S}$. This case follows as for Proposition 5.

Proof of Theorem 2. We prove by structural induction on P that all conditions in Definition 9 are satisfied. First of all, note that the language of orchestrators we rely on does not diverge, hence consistency condition (3) is trivially satisfied in all cases

- $\mathbf{P} = \mathbf{0}$: Conditions (1), (2) hold trivially since P has no reductions. As far as condition(4) is concerned, note that $\sigma = 0$ and $\sigma \Downarrow R$ implies $R = \emptyset \subseteq A$ for any A.
- $\mathbf{P} = \lambda.\mathbf{P}'$. If $\lambda = \tau$ then $P = \tau P'$. The only possible type for P (derived by using rule (TAU)) is σ with $\vdash P' : \sigma$ and clearly $\sigma \sqsubseteq \sigma$ and therefore condition (1) holds. Moreover, conditions (2) and (4) trivially hold. Let $\lambda \neq \tau$. Then, condition (1) trivially hold. As regards to condition (2), note that $\vdash P : \sigma$ with $\sigma = \lambda.\sigma'$ and $\vdash P' : \sigma'$. Consequently, $\sigma(\lambda) = \sigma'$ and condition (2) holds. As condition (4) is concerned, note that $P \Downarrow R$ implies $R = \{\lambda\} = \{\lambda \mid P \overset{\lambda}{\to}\}$.
- $\mathbf{P} = \Sigma_{\mathbf{i}\in\mathbf{I}} \lambda_\mathbf{i}.\mathbf{P_i} + \Sigma_{\mathbf{j}\in\mathbf{J}} \tau.\mathbf{Q_j}$. From typing rule (SUM) we have that $\vdash P : \sigma$ with $\sigma = \Sigma_{i\in I}\lambda_i.\sigma_i \oplus \bigoplus_{j\in J} \rho_j$. Condition (1): If $P \overset{\tau}{\to} P'$ then there exists some $k \in J$ such that $P' = Q_k$ and $\vdash P' : \rho_k$ with $\vdash Q_k : \rho_k$. Note that $\sigma = \rho_k \oplus \tau$ for some τ. Consequently, $\sigma = \rho_k \oplus \tau \sqsubseteq \rho_k = \sigma'$. Condition (2): If $P \overset{\lambda}{\to} P'$ with $\lambda \neq \tau$, then there exists some $k \in I$ such that $P' = P_k$ and $\lambda_k = \lambda$ and $\vdash P' : \sigma_k$. Consequently, $\sigma(\lambda) = \sigma_k \oplus \tau$ for some τ. Hence, $\sigma(\lambda) \sqsubseteq \sigma'$. As far as condition (4) is concerned, note that $P \overset{\tau}{\not\to}$ implies $J = \emptyset$.Then $\sigma = \Sigma_{i\in I}\lambda_i.\sigma_i$ then $\sigma \Downarrow R$ implies $R = \{\lambda_i | i \in I\} = \{\lambda \mid P \overset{\alpha}{\to}\}$.
- $\mathbf{P} = \mathbf{P_1}|\mathbf{P_2}$. Condition (1), if $P \overset{\tau}{\to} P'$ then either $P_1 \overset{\tau}{\to} P_1'$ or $P_2 \overset{\tau}{\to} P_2'$. If $P_1 \overset{\tau}{\to} P_1'$ then σ is an internal choice containing a subterm σ' where $\vdash P_1'|P_2 : \sigma'$. Consequently, $\sigma' \sqsubseteq \sigma$. The case $P_2 \overset{\tau}{\to} P_2'$ is analogous. For condition (2), note that either $P_1 \overset{\lambda}{\to} P_1'$ or $P_2 \overset{\lambda}{\to} P_2'$ and $\lambda \neq \tau$. The proof follows as for condition (1). As regards to condition (3), note that neither $P_1 \overset{\tau}{\to}$ nor $P_1 \overset{\tau}{\to}$. Hence, $\sigma = (\Sigma_{\lambda_i\in V}\lambda_i.\sigma_i + \Sigma_{\beta_j\in V}\beta_j.\rho_j)$ where $P \overset{\lambda_i}{\longrightarrow} P_i$ and $Q \overset{\beta_j}{\longrightarrow} Q_j$

  Therefore $\sigma \Downarrow R$ implies $R = \{\lambda | P \overset{\lambda}{\to}\}$.
- $\mathbf{P} = \mathtt{if}\ \mathbf{m} = \mathbf{n}\ \mathtt{then}\ \mathbf{P_1}\ \mathtt{else}\ \mathbf{P_2}$. There are two cases $\square \in \{m,n\}$ and $\square \notin \{m,n\}$. Assume $\square \in \{m,n\}$. By rule (COND1), $\sigma = \sigma_1 \oplus \sigma_2$ with $\vdash P_1 : \sigma_1$ and $\vdash P_2 : \sigma_2$. As far as condition (1) is concerned, $P \overset{\tau}{\to} P'$ when either $P_1 \overset{\tau}{\to} P_1'$ or $P_2 \overset{\tau}{\to} P_2'$. Let $P_1 \overset{\tau}{\to} P_1'$ with $\vdash P_1' : \sigma_1'$ and $\sigma_1 \sqsubseteq \sigma_1'$ by inductive hypothesis. Therefore, $\sigma = \sigma_1 \oplus \sigma_2 \sqsubseteq \sigma_1'$. The case $P_2 \overset{\tau}{\to} P_2'$ follows analogously. For condition (2), the proof follows analogously to condition 1. In respect to condition (3), note that $\sigma \Downarrow R$ implies that either $\sigma_1 \Downarrow R$ or $\sigma_2 \Downarrow R$. By inductive hypothesis, we know that $\sigma_1 \Downarrow R$ implies $R \subseteq \{\lambda | P_1 \overset{\lambda}{\to}\}$ and $\sigma_2 \Downarrow R$ implies $R \subseteq \{\lambda | P_2 \overset{\lambda}{\to}\}$. Hence, $R \subseteq \{\lambda | P_1 \overset{\lambda}{\to}\} \cup \{\lambda | P_2 \overset{\lambda}{\to}\}$. It is easy to see that $P \overset{\lambda}{\to}$ if and only if $P_1 \overset{\lambda}{\to}$ or $P_1 \overset{\lambda}{\to}$. The cases for $\square \notin \{m,n\}$ follows analogously by noting that σ is either σ_1 or σ_2 depending on whether $m = n$ or $m \neq n$ hold.

Proof of Proposition 8. We only prove the first case above; the second case is similar. From $A_V[Q] \xrightarrow{\alpha} A_V[Q']$ we have that $Q \xrightarrow{\beta} Q'$ with $\beta_{|V} = \alpha$, being $\beta_{|V}$ defined as the expected counterpart of $\lambda_{|V}$. By Definition 1, there exist M, λ, R and $\sigma \models M$ such that $Q \xrightarrow{M,\lambda} R$ and $\lambda\sigma = \beta$ and $R\sigma = Q'$. P and Q are closed, hence $bn(\lambda) \cap fn(P,Q,M) = \emptyset$. Since, $P \propto^V Q$ there exists a M-decomposition D such that $\forall M' \in D$, $P \xrightarrow{N',\lambda'} P''$ with $M' \Rightarrow N'$, $\lambda_{|V} = \lambda'$, and there exists some simulation-based abstraction relation $\mathscr{R}^V_M$ such that $P'' \mathscr{R}^{V \cup vn(\lambda)}_{M'} R$. Since $\sigma \models M$ and D is a M-decomposition, there exists at least one $M_i \in D$ such that $\sigma \models M_i$ (and hence $\sigma \models N'$). By Definition 1, $P \xrightarrow{\lambda'\sigma} P''\sigma$. There are two cases:

- $\lambda' = \tau$ or $\lambda' = \overline{x}\langle\widetilde{a}\rangle$: In both cases, λ and λ' are closed. Hence, $\lambda' = \lambda'\rho$ for any substitution ρ. Since, $\beta = \lambda\sigma = \lambda$, we have that $\alpha = \beta_{|V} = \lambda_{|V} = \lambda' = \lambda'\sigma$.
  It remains to show that $P''\sigma \propto^V Q'$ with $Q' = R\sigma$. Since $bn(\lambda) = \emptyset$, we have $P'' \mathscr{R}^V_{M'} R$. Also note that R and P'' are closed because Q and P are closed. Hence, $Q' = R\sigma = R$, $P'' = P''\sigma$ and $P''\sigma \mathscr{R}^V_{M'} Q'$.
- $\lambda' = x(\widetilde{v})$: Since $\lambda' = \lambda_{|V}$ is an input action, we have that λ is an input action and both λ' and λ have the same subject x that belongs to V. Consequently, $\lambda_{|V} = \lambda$. Then, $\lambda' = \lambda_{|V} = \lambda$, and consequently $\lambda'\sigma = \lambda\sigma = \beta$. Since, β is an input action whose subject is in V, $\beta_{|V} = \beta$. Consequently, $\lambda'\sigma = \lambda\sigma = \beta_{|V} = \alpha$. It remains to show that $P''\sigma \propto^V Q'$ with $Q' = R\sigma$. We know that $P'' \mathscr{R}^{V \cup vn(\lambda)}_{M'} R$. Since λ is an input action $\widetilde{v} \cap fn(M') = \emptyset$, hence $\sigma \models M'$. By Proposition 12, $P''\sigma \mathscr{R}^V_{M'} R\sigma$. Since $P''\sigma$ and $R\sigma$ are closed, $P''\sigma \propto^V R\sigma$ holds by Proposition 10.