# A Java Inspired Semantics for Transactions in SOC
## Extended Paper

Laura Bocchi and Emilio Tuosto

Department of Computer Science, University of Leicester, UK

**Abstract.** We propose a formal semantics for distributed transactions inspired by the attribute mechanisms of the Java Transaction API. Technically, we model services in a process calculus featuring transactional scope mechanisms borrowed from the so called *container-managed* transactions of Java. We equip our calculus with a type system for our calculus and show that, in well-typed systems, it guarantees absence of run-time errors due to misuse of transactional mechanisms.

## 1 Introduction

The *Service-Oriented Computing* (SOC) paradigm envisages distributed systems as loosely-coupled computational entities which dynamically discover each other and bind together. Although appealing, SOC has imposed to re-think, among other classic concepts, the notion of transaction. The long lasting and cross-domain nature of SOC makes typically unfeasible to adopt ACID transactions, which are implemented by locking the involved resources. The investigation of formal semantics of SOC transactions (often referred to as long-running transactions) has been a topic of focus in the last few years (see § 8 for a non-exhaustive overview). Central to this investigation is the notion of *compensation* (a weaker and "ad hoc" version of the classic rollback of database systems) which has mainly been studied in relation to mechanisms of failure propagation.

In this paper we address an orthogonal topic, namely the semantics of dynamic reconfiguration of transactions in SOC which, to the best of our knowledge, has not been explicitly considered. In SOC, the configuration of a system can change at each service invocation to include a new instance of the service in the ongoing computation. There is still a lack of agreement on how the run-time reconfiguration should affect the relationships between existing and newly created transactional scopes. To illustrate the main problems, we consider the following example:

$$\langle \texttt{invoke}\ s.P \mid (\!|C|\!) \rangle \qquad \text{with } s \text{ implemented as } Q \tag{1}$$

where a process in a transactional scope (represented by the angled brackets) with compensation $C$ invokes a service $s$ and then behaves like $P$; the invocation triggers a (possibly remote) instance $Q$ of the service $s$. Should the system in (1) evolve to a transactional scope that includes $Q$ (i.e., $\langle P \mid Q \mid (\!|C|\!) \rangle$)? Should instead $Q$ be running in a different scope (i.e., $\langle P \mid (\!|C|\!) \rangle \mid \langle Q \rangle$)? Or should $Q$ be executed outside any transactional scope (i.e., $\langle P \mid (\!|C|\!) \rangle \mid Q$) or else raise an exception triggering the compensation $C$? Notice that each alternative is valid and has an impact on failure propagation.

Enterprise Java Beans (EJB) promote *Container Managed Transactions* (CMT) as a mechanism to control dynamic reconfigurations. We take inspiration from the EJB mechanism and adapt it to SOC transactions. A *container* can be used to publish objects and can specify:

- the transactional modality of method calls (e.g., "*calling the method* `fooBar` *from outside a transactional scope throws an exception*"),
- how the scope of transactions dynamically reconfigure (e.g., "`fooBar` *is always executed in a newly created transactional scope*").

A limitation of CMT is that it only permits to declare transactional modalities for the methods to be invoked and does not allow invokers to specify their own requirements on the needed transactional support. On the contrary service invocations are resolved at run-time and different providers may publish different implementations of a service. Hence, it is natural to give the invoker the opportunity to express some requirements on the transactional behaviour of the invoked services. For instance, in (1) the invocation to $s$ may require that $Q$ must be executed in the same transactional scope of $P$.

We do not aim to provide a semantics for CMT but rather investigate how CMT could be borrowed to address the issues described above for SOC transactions. We promote some CMT inspired primitives for SOC which allow invokers (and not just callees) to specify their own transactional requirements. Furthermore, we give a typing discipline to ensure that invocations do not yield run-time errors due to the incompatibility of the transactional modalities required by callers and those guaranteed by callees.

Our main contributions are

1. a semantics to specify dynamic reconfiguration of SOC transactions inspired by the CMT mechanisms of EJB; namely, we introduce a CCS-like process calculus called ATc (after *Attribute-based Transactional calculus*)
2. a type system that guarantees that no error will occur for a method invocation due to the incompatibility of the transactional scopes of caller and callee
3. a methodology for designing SOC transactions based on our typing discipline.
4. a theory of testing equivalence for ATc, based on the notion of observation and testing equivalence in [15].
5. a few results obtained applying the provided testing environment. Specifically, we compare the behaviour of processes (1) with respect to their nesting in transactional scopes and (2) that specify different transactional attributes. As for (2), different attributes cause different behaviour in general but under some conditions some of them are interchangeable (Theorem 2). Also, Theorem 2 allows a caller to specify a larger set of attributes (by including all the interchangeable ones) and possibly increases the probability of finding a matching service. Finally, we prove that a testing preorder can be established for a non-trivial subclass of systems (Proposition 6) contrary to the general case (Proposition 5).

**Synopsis** The transactional mechanisms of EJB are summarised in § 2. The syntax and semantics of ATc are introduced in § 3. The typing discipline of ATc is in § 4. In § 5 we give a gist of how our type system can be used to design systems correct wrt dynamic reconfigurations of transactions. Conclusions and related work are discussed in § 8.

## 2 EJB Transactional Attributes

Roughly, a *Java bean* can be thought of as a Java object amenable to be executed in a specialised run-time environment called *container* (see e.g., [14, 19]). An EJB container provides standard functionalities to components; in particular, a container is responsible for the life-cicle of a bean and makes it accessible to other components by binding it to a naming service[1].

For the sake of this paper, we focus on the transactional mechanisms offered by EJB containers. Specifically, we consider *Container Managed Transactions* (CMT) whereby a container associates each method of a bean with a *transactional attribute* specifying the modality of reconfiguring transactional scopes. We denote the set of EJB transactional attributes as

$$\text{(EJB Transactional Attributes)} \qquad \mathcal{A} \stackrel{def}{=} \{\mathtt{m}, \mathtt{s}, \mathtt{n}, \mathtt{ns}, \mathtt{r}, \mathtt{rn}\}$$

where, following the EJB terminology, `m` stands for *mandatory*, `s` for *supported*, `n` for *never*, `ns` for *not supported*, `r` for *requires*, and `rn` for *requires new*.

The intuitive semantics of EJB attributes $\mathcal{A}$ (ranged over by $a, a_1, a_2, \ldots$) is illustrated in Figure 1 where each row represents the behaviour of one transactional attribute and shows how the transactional scope (represented by a rectangular box) of the caller (represented by a filled circle) and callee (represented by an empty circle) behave upon invocation. The first two columns of Figure 1 represent, respectively, invocations from

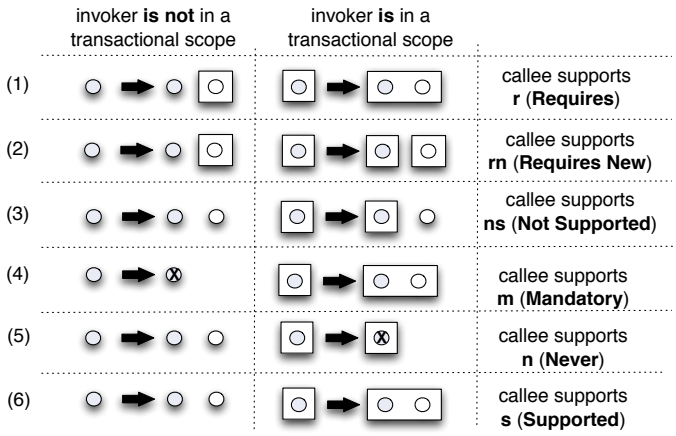



**Fig. 1.** EJB transactional attributes synopsis

outside and from within a transactional scope. More precisely, (1) a callee supporting `r` is always executed in a transactional scope which happens to be the same as the caller's if the latter is already running in a transactional scope; (2) a callee supporting `rn` is

[1] `http://docs.sun.com/app/docs/doc/819-3658/ablmw?a=view`

always executed in a new transactional scope; (3) a callee supporting `ns` is always executed outside a transactional scope; (4) the invocation to a method supporting `m` fails if the caller is not in a transactional scope (first column of the fourth row in Figure 1), otherwise the method is executed in the transactional scope of the caller; (5) the invocation to a method supporting `n` is successful only if the caller is outside a transactional scope, and it fails if the caller is running in a transactional scope (in this case an exception is triggered in the caller); (6) a method supporting `s` is executed inside (resp. outside) the caller's scope if the caller is executing in (resp. outside) a scope.

In this paper, we adapt the transactional model of EJB to the context of SOC, where each provider can be thought of as a container specifying a number of services together with their transactional attribute. A transactional attribute declares whether a published service must or must not be executed within a transactional scope and the modality of dynamic reconfiguration of the transactional scope (e.g., whether a new scope has to be created, how the scope of the invoking party has to be extended, etc.). We formally model the behaviour illustrated in Figure 1 by embedding EJB attributes in a simple process calculus to give a general model for SOC[2]. Hereafter, according to this interpretation, the terms *service provider* and *container* will be used interchangeably.

## 3 Attribute-based Transaction calculus (ATc)

The ATc calculus is built on top of two layers; *processes* (§ 3.1) and *systems* (§ 3.2). The former specify how communication takes place in presence of (nested) transactional scopes while the latter provide a formal framework for defining and invoking transactional services and the run-time reconfiguration of the transactional scopes.

### 3.1 ATc processes

An ATc process is a CCS-like process with three additional capabilities: *service invocations*, *transactional scoping*, and *compensation installation*. Let $\mathcal{S}$ and $\mathcal{N}$ be two countably infinite and disjoint sets of names for *service* and *channel*, respectively.

**Definition 1.** *The set* ATc processes $\mathcal{P}$ *is defined by following grammar:*

| | | | | | |
|---|---|---|---|---|---|
| $P, Q ::=$ | $0$ | *empty process* | $\pi ::=$ | $x$ | *input* |
| | $\nu x\, P$ | *channel restriction* | | $\overline{x}$ | *output* |
| | $P \mid Q$ | *parallel* | | | $A \subseteq \mathcal{A}$ |
| | $!P$ | *replication* | | | $s, s', \ldots$ *range over* $\mathcal{S}$ |
| | $s \triangleleft A.P$ | *service invocation* | | | $x, y, z, \ldots$ *range over* $\mathcal{N}$ |
| | $\langle P \mid (\!\lvert Q \rvert\!) \rangle$ | *transactional scope* | | | $u$ *ranges over* $\mathcal{S} \cup \mathcal{N}$ |
| | $\pi[\![Q]\!].P$ | *compensation installation* | | | |

*Restriction $\nu x\, P$ binds $x$ in $P$ and the sets of* free *and* bound *channels of $P \in \mathcal{P}$ are defined as usual and respectively denoted by $\mathsf{fc}(P)$ and $\mathsf{bc}(P)$.*

[2] We refer to the service-oriented paradigm in a technology-agnostic way, abstracting from its actual realisations (e.g., the Web Service Architecture).

The standard process algebraic syntax is adopted for idle process, restriction, parallel composition, and replication. Process $s \triangleleft A.P$ invokes a service $s$ required to support a transactional attributes in $A \subseteq \mathcal{A}$; a transactional scope $\langle P \mid (\!| Q |\!) \rangle$ consists of a running process $P$ and a compensation $Q$ (confined in the scope) executed only upon failure; $\pi[\![Q]\!].P$ executes $\pi$ and installs the compensation $Q$ in the enclosing transactional scope then behaves as $P$. Service definition and invocation are dealt with in § 3.2.

**Definition 2.** *The* structural congruence $\equiv \subseteq \mathcal{P} \times \mathcal{P}$*, is the smallest equivalence relation containing $\alpha$-renaming, the monoidal axioms for $\mid$ and $\mathsf{0}$, and satisfying:*

$$!P \mid P \equiv !P \quad \langle \mathsf{0} \mid (\!| Q |\!) \rangle \equiv \mathsf{0} \equiv (\!| \mathsf{0} |\!) \quad (\!| P |\!) \mid (\!| Q |\!) \equiv (\!| P \mid Q |\!)$$

$$\textit{if } P \equiv Q \textit{ then } \langle P \rangle \equiv \langle Q \rangle \textit{ and } (\!| P |\!) \equiv (\!| Q |\!)$$

$$\nu x\, \langle P \rangle \equiv \langle \nu x\, P \rangle \quad \nu x\, \nu y\, P \equiv \nu y\, \nu x\, P \quad \nu x\, \mathsf{0} \equiv \mathsf{0} \quad \nu x\, (P \mid Q) \equiv (\nu x\, P) \mid Q, \textit{ if } x \notin \mathsf{fc}(Q)$$

*Hereafter, $\pi.P$ stands for $\pi[\![Q]\!].P$ when $Q \equiv \mathsf{0}$ and trailing occurrences of $\mathsf{0}$ are omitted.*

In ATc, transactional scopes can be nested up to an arbitrary level. The fact that a process is inside a transactional scope does not alter its communication capabilities, since we assume that transactional scopes influence the behaviour of processes only in case of failure. To model the semantics of communications we use *contexts*[3].

**Definition 3.** *A* context *is a term generated by the following productions:*

$$\mathbf{C}[\square] ::= \square \;\mid\; \mathsf{0} \;\mid\; \langle \square \mid P \mid (\!| Q |\!) \rangle \;\mid\; P \mid \mathbf{C}[\square] \;\mid\; \mathbf{C}[\square] \mid P$$

*A context $\mathbf{C}[\square]$ is* scope-avoiding *(s-a, for short) if there are no $P, Q \in \mathcal{P}$ and context $\mathbf{C}'[\square]$ such that $\mathbf{C}[\square] = \mathbf{C}'[\langle \square \mid P \mid (\!| Q |\!) \rangle]$.*

Definition 3 does not consider $\nu x\ \mathbf{C}[\square]$ to avoid name capture while prefix contexts $\alpha.\mathbf{C}[\square]$ (where $\alpha$ is either of the prefixes of ATc) are ruled out as they prevent inner reductions. The semantics of ATc is defined by means of two reduction relations, one (Definition 4) for process communication and the other (Definition 6) for service invocations (and, correspondingly, reconfigurations of transactional scopes).

**Definition 4.** *The* reduction relation of ATc processes *is the smallest relation $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$ closed under the following axioms and rules:*

(p1) $\mathbf{C}[\langle \pi[\![Q]\!].P \mid (\!| R |\!) \rangle] \mid \mathbf{C}'[\langle \bar{\pi}[\![Q']\!].P' \mid (\!| R' |\!) \rangle] \rightarrow \mathbf{C}[\langle P \mid (\!| R \mid Q |\!) \rangle] \mid \mathbf{C}'[\langle P' \mid (\!| R' \mid Q' |\!) \rangle]$

(p2) $\mathbf{C}[\langle \pi[\![Q]\!].P \mid (\!| R |\!) \rangle] \mid \mathbf{C}'[\bar{\pi}[\![Q']\!].P'] \rightarrow \mathbf{C}[\langle P \mid (\!| R \mid Q |\!) \rangle] \mid \mathbf{C}'[P'], \quad \textit{if } \mathbf{C}'[\square] \textit{ is s-a}$

(p3) $\mathbf{C}[\pi[\![Q]\!].P] \mid \mathbf{C}'[\bar{\pi}[\![Q']\!].P'] \rightarrow \mathbf{C}[P] \mid \mathbf{C}'[P'], \quad \textit{if } \mathbf{C}[\square] \textit{ and } \mathbf{C}'[\square] \textit{ are s-a}$

(p4) $\dfrac{P \rightarrow P'}{P \mid R \rightarrow P' \mid R}$ (p5) $\dfrac{P \rightarrow P'}{\nu x\, P \rightarrow \nu x\, P'}$ (p6) $\dfrac{P \equiv P' \rightarrow Q' \equiv Q}{P \rightarrow Q}$

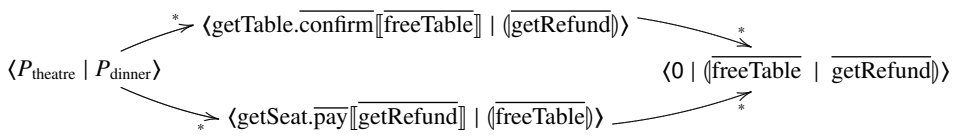
[3] Other and more standard techniques could have been used (e.g., LTS); however, contexts enable us to easily define the semantics of communication and service invocation of ATc.

Notice that sender and receiver synchronise regardless the relative nesting of transactional scopes. As in [11], when communication actions are executed compensations are installed in parallel to the other compensations of the enclosing transactional scope if any, otherwise they are discarded. In case of failure, only the actions executed before the failure are compensated, as illustrated by Example 1.

*Example 1.* Consider the transactional scope $P_{\text{bookNight}} = \langle P_{\text{theatre}} \mid P_{\text{dinner}} \rangle$ where:

$$P_{\text{theatre}} = \overline{\text{askSeat}}.\text{getSeat}.\overline{\text{pay}}[\![\overline{\text{getRefund}}]\!] \qquad P_{\text{dinner}} = \overline{\text{askTable}}.\text{getTable}.\overline{\text{confirm}}[\![\overline{\text{freeTable}}]\!]$$

Action $\overline{\text{getRefund}}$ compensates $\overline{\text{pay}}$ and action $\overline{\text{freeTable}}$ compensates $\overline{\text{confirm}}$. The process dynamically installs the compensations of its actions. The two executions




show that different compensations may be executed in case of failure. ◇

### 3.2 ATc systems

The semantics of transactional scoping of service invocations is given at the level of *systems* (Definition 5). Systems can be thought of as an abstraction for EJB and consist of processes wrapped by *containers* defined as a partial finite maps $\gamma : \mathcal{S} \to \mathcal{A} \times \mathcal{P}$; containers assign a transactional attribute and a process (the "body") to service names. When defined, $\gamma(s) = (a, P)$ ensures that, if invoked in $\gamma$, the service $s$ supports the attribute $a$ and activates an end-point that executes as $P$.

**Definition 5.** *A* system *in ATc is a pair $\Gamma \vdash P$ where the* environment *$\Gamma$ is a set of containers and is derived by the productions in Definition 1 augmented with $P ::= \texttt{err}$ to represent* erroneous processes*. Also, the following axioms*

$$!\texttt{err} \equiv \texttt{err} \qquad \nu x\, \texttt{err} \equiv \texttt{err} \qquad \langle \texttt{err} \mid (\!|Q|\!) \rangle \equiv \texttt{err}$$

*extend the congruence relation to erroneous processes.*

Given $A \subseteq \mathcal{A}$, $P \in \Gamma(s, A)$ shortens $\exists \gamma \in \Gamma\ \exists a \in A\ :\ \gamma(s) = (a, P)$ and $P \in \Gamma(s, \{a\})$ is abbreviated as $P \in \Gamma(s, a)$. Hereafter, we use $P$, $Q$ to range over both $\mathcal{P}$ and erroneous processes. We rule out terms where compensations contain `err`; basically, `err` represent a run-time error and cannot be used by the programmer. A service invocation is *transactional* (resp. *non-transactional*) if it is (resp. not) executed a transaction scope.

Definition 6 formalises the informal presentation in Figure 1 of the CMT mechanisms which are rendered in SOC by allowing environments $\Gamma$ to offer different implementations of the same service possibly with different attributes. This results in a non-deterministic semantics where one of several possible reductions is chosen.

**Definition 6.** *The* reduction relation of ATc systems *is the smallest relation $\rightsquigarrow$ closed under the following rule and axioms of Figure 2 where $\mathbf{C}[\square] \neq 0$ and $\mathbf{C}[\square]$ is s-a in $(\texttt{ntx}1 \div 3)$.*

$$\texttt{(ntx1)}\quad \Gamma \vdash \mathbf{C}[s \,\&\!\leftarrow A.P] \rightsquigarrow \Gamma \vdash \mathbf{C}[\texttt{err}] \qquad \texttt{m} \in A$$

$$\texttt{(ntx2)}\quad \Gamma \vdash \mathbf{C}[s \,\&\!\leftarrow A.P] \rightsquigarrow \Gamma \vdash \mathbf{C}[P] \mid R \qquad R \in \Gamma(s, \{\texttt{s}, \texttt{n}, \texttt{ns}\} \cap A)$$

$$\texttt{(ntx3)}\quad \Gamma \vdash \mathbf{C}[s \,\&\!\leftarrow A.P] \rightsquigarrow \Gamma \vdash \mathbf{C}[P] \mid \langle R \rangle \qquad R \in \Gamma(s, \{\texttt{r}, \texttt{rn}\} \cap A)$$

$$\texttt{(tx1)}\quad \frac{P = \mathbf{C}[\langle s \,\&\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q|\!) \rangle] \qquad \mathsf{bc}(P) \cap \mathsf{fc}(R) = \emptyset}{\Gamma \vdash P \rightsquigarrow \Gamma \vdash \mathbf{C}[\langle P_1 \mid P_2 \mid R \mid (\!|Q|\!) \rangle]} \qquad R \in \Gamma(s, \{\texttt{m}, \texttt{s}, \texttt{r}\} \cap A)$$

$$\texttt{(tx2)}\quad \Gamma \vdash \mathbf{C}[\langle s \,\&\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q|\!) \rangle] \rightsquigarrow \Gamma \vdash \mathbf{C}[Q] \qquad \texttt{n} \in A$$

$$\texttt{(tx3)}\quad \Gamma \vdash \mathbf{C}[\langle s \,\&\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q|\!) \rangle] \rightsquigarrow \Gamma \vdash \mathbf{C}[\langle P_1 \mid P_2 \mid (\!|Q|\!) \rangle] \mid R \qquad \texttt{ns} \in A \wedge R \in \Gamma(s, \texttt{ns})$$

$$\texttt{(tx4)}\quad \Gamma \vdash \mathbf{C}[\langle s \,\&\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q|\!) \rangle] \rightsquigarrow \Gamma \vdash \mathbf{C}[\langle P_1 \mid P_2 \mid (\!|Q|\!) \rangle] \mid \langle R \rangle \qquad \texttt{rn} \in A \wedge R \in \Gamma(s, \texttt{rn})$$

$$\texttt{(s-p)}\quad \frac{P \to P'}{\Gamma \vdash P \rightsquigarrow \Gamma \vdash P'}$$

**Fig. 2.** Semantics of ATc

Axioms (ntx1÷3) rule non-transactional invocations; (ntx1) states that an invocation results in an error when a service supporting attribute m is required[4]; when a non-transactional invocation is made to a service supporting either s, or n, or ns, by (ntx2), the end-point of the service is executed in parallel with the continuation of the caller; finally by (ntx3), the end-point of a service supporting r or rn will be executed in a new scope (initially with idle compensation).

Axioms (tx1÷4) determine how transactional invocations modify the scope; by (tx1), the end-point of the service is executed in the same scope of the caller when the requested attribute is m, s, or r; instead by (tx2), transactional invocations to a service supporting n yields a failure which triggers the compensation of the caller; by (tx3) a transactional invocation requesting ns will let the service end-point to run outside the caller's scope; finally, (tx4) states that a transactional invocation requesting rn will let the service end-point to run in a new scope with idle compensation.

Rule (s-p) lifts process reduction relation to systems.

Communication failures occurring within transactional scopes trigger compensations while those occurring outside result in an error. Formally, this is modelled within the theory of testing [15] for ATc given in § 6.

### 3.3 Some examples of failing invocations

The following examples motivate the need of a disciplined use of transactional attributes. The typing system presented in § 4 ensures that a well-typed process will incur

[4] Axiom (ntx1) may seem odd as it introduces an error even if $\Gamma$ may offer a service supporting other attributes in $A$. An actual implementation may in fact select more suitable services with an appropriate negotiation in the search phase. Here, more simply, we define the conditions to correctly use attributes avoiding errors in *any* possible environment; therefore (ntx1) models the worst case scenario. As shows in § 4, in well-typed processes, invocations requiring m never occur in s-a contexts.

in errors due to the fact that the attributes required by an invoker do not match those guaranteed by the service.

*Example 2.* Let $P_{\text{bookTheatre}} = \langle s_{\text{tickets}} \,\&\!\!\leftarrow\, \{\mathtt{m}\}.P_{\text{theatre}} \mid (\!|\, s_{\text{compensate}} \,\&\!\!\leftarrow\, \{\mathtt{m}\} \,|\!)\rangle$ be a process that invokes $s_{\text{tickets}}$ and behaves as $P_{\text{theatre}} = \overline{\text{askSeat}}.\text{getSeat}.\overline{\text{pay}}[\![\overline{\text{getRefund}}]\!]$. If a communication of $P_{\text{theatre}}$ fails (i.e., the left-most axiom in (2) is applied), then the compensation is executed outside a transactional scope. Therefore, the non-transactional invocation to $s_{\text{compensate}}$ will result in an error. ◇

*Example 3.* Let $P_{\text{theatre}}$ as in Example 2 and consider

$$P_{\text{bookTheatre}} = s_{\text{tickets}} \,\&\!\!\leftarrow\, \{\mathtt{m}, \mathtt{s}, \mathtt{n}, \mathtt{ns}, \mathtt{r}, \mathtt{rn}\}.P_{\text{theatre}} \qquad P_{\text{tickets}} = \text{askSeats}.\overline{\text{getSeats}}.s_{\text{bank}} \,\&\!\!\leftarrow\, \{\mathtt{m}\}$$

The non-transactional invocation $s_{\text{tickets}}$ in a $\Gamma$ for which $P_{\text{tickets}} \in \Gamma(s_{\text{tickets}}, \mathtt{s})$ causes $P_{\text{tickets}}$ to run outside a transactional scope; hence, invoking $s_{\text{bank}}$ leads to an error. ◇

A provider must guarantee that none of its services yield errors; namely, the execution of (the body of) a service in any context resulting from its supported attributes should be safe. For instance, since $s_{\text{tickets}}$ in Example 3 supports $\mathtt{s}$, the execution $P_{\text{tickets}}$ should be safe regardless it will run inside or outside a transactional scope. In fact, whether or not $P_{\text{tickets}}$ will be running in a scope depends on the caller.

## 4 A Type System for Transactional Services

This section yields a type system for ATc that can determine if a system may fail for a service invocation due to misuse of the transactional attributes. We give an algebra of types (§ 4.1), then define a type system for ATc (§ 4.2), and finally we give a suitable notion of well-typedness for ATc systems (§ 4.3) which is preserved by the reduction relation (Theorem 1) and ensures error-freedom (Corollary 1). All the proofs are reported in Appendix A.

### 4.1 Types for ATc

Our types record which transactional attributes may be required/supported in service invocations of processes. Basically, for each possible invocation, a type specifies if it is transactional or not and which transactional attributes are declared for the invocation.

**Definition 7.** *Let $I \subseteq \{\mathfrak{i}, \mathfrak{o}\} \times \mathcal{A}$ where labels $\mathfrak{i}$ and $\mathfrak{o}$ are the* transactional modalities *used to keep track of transactional and non-transactional invocations, respectively. Types are defined as*

$$(Types) \quad t ::= 0 \;\big|\; (I, t, t)$$

Let $P \triangleright t$ state that $P \in \mathcal{P}$ has type $t$. If $P \triangleright 0$ then $P$ does not make any invocations; if $P \triangleright (I, t_c, t_u)$,

- $I$ records the transactional modality/attribute pairs of the service invocations of $P$;
- $t_c$ collects the transactional modality/attribute pairs relative to the service invocations in the compensations of the transactional scopes of $P$;

$t_u$ yields modality/attribute pairs for the invocations in the compensation installation prefixes[5] of $P$;

*Example 4.* Consider $P_2 = s \leftarrowtail A.y[\![P_1]\!]$ with $P_1 \triangleright t_1$. As more clear in § 4.2, $P_2 \triangleright t_2 = (\{\mathfrak{o}\} \times A, \mathsf{0}, t_1)$. In fact, the invocation in $P_2$ is non-transactional and the third component of $t_2$ is $t_1$ as $P_1$ is used to compensate prefix $y$. ◇

Types of processes become more complex in presence of nested scopes.

*Example 5.* Take the process $P_3 = \langle P_2 \mid (\!| s' \leftarrowtail A' |\!) \rangle \mid \langle\langle P_2 \mid (\!| s' \leftarrowtail A'' |\!) \rangle\rangle$, where $P_2$ is defined in Example 4. The type of $P_3$ is

$$t_3 = (\{\mathfrak{i}\} \times (A \cup A''), (\{\mathfrak{o}\} \times A', \mathsf{0}, \mathsf{0}), \mathsf{0})$$

In fact, the invocations in $P_2$ and in the nested compensation in the rightmost scope of $P_3$ will be transactional; therefore the first component of $t_3$ is $\{\mathfrak{i}\} \times (A \cup A'')$. Moreover, the leftmost scope of $P_3$ may possibly have a non-transactional invocation (thereby the second component of $t_3$). ◇

The next example illustrates the installation of a non-trivial compensation.

*Example 6.* The type of $P_4 = \overline{z}[\![s_1 \leftarrowtail A_1]\!].\langle \overline{z}[\![s_1 \leftarrowtail A_1]\!] \mid (\!| s_2 \leftarrowtail A_2.z[\![s_3 \leftarrowtail A_3]\!] |\!) \rangle$ is

$$t_4 = (\mathsf{0}, (\{\mathfrak{o}\} \times (A_1 \cup A_2), \mathsf{0}, \{\mathfrak{o}\} \times A_3), \{\mathfrak{o}\} \times A_1)$$

In fact, $P_4$ does not invoke services but installs compensations that do so. Observe that the third component of $t_4$ corresponds to the first installation of $P_4$, while the second component of $t_4$ is the type of the scope occurring in $P_4$. ◇

It is convenient to treat types as binary trees whose nodes are labelled with subsets of $\{\mathfrak{i}, \mathfrak{o}\} \times \mathcal{A}$. More precisely, the type $(I, t_c, t_u)$ can be represented as a tree where the root is labelled $I$, $t_c$ is the left child, and $t_u$ is the right child ($\mathsf{0}$ is the empty tree which is conventionally labelled with the empty set). The operators $\_^{\leftarrowtail}$, $\_^{\downarrow}$, and $\_^{?}$ are used to "traverse" types and $\_ \oplus \_$ to "sum" them as per the following definitions:

$$\mathsf{0}^{\leftarrowtail} = \emptyset, \qquad \mathsf{0}^{\downarrow} = \mathsf{0}^{?} = \mathsf{0} \qquad (I, t_c, t_u)^{\leftarrowtail} = I, \qquad (I, t_c, t_u)^{\downarrow} = t_c, \qquad (I, t_c, t_u)^{?} = t_u$$
$$\mathsf{0} \oplus t = t, \qquad (I, t_c, t_u) \oplus (I', t'_c, t'_u) = (I \cup I', t_c \oplus t'_c, t_u \oplus t'_u)$$

We assume that $\_ \oplus \_$ has lower precedence than unary operators.

Propositions 1 and 2 will be tacitly used in the proofs of the lemmas and Theorem 1.

**Proposition 1.** *The operator $\_ \oplus \_$ is idempotent, associative and commutative.*

**Proposition 2.** *Operators $\_^{\leftarrowtail}$, $\_^{\downarrow}$, and $\_^{?}$ distribute over $\_ \oplus \_$ and $(t_1 \oplus t_2)^{\leftarrowtail} = {t_1}^{\leftarrowtail} \cup {t_2}^{\leftarrowtail}$.*

[5] By Definition 6 compensations vanish for synchronisations outside transactional scopes.

### 4.2 Typing ATc

This section introduces a typing system for ATc. We recall that the ATc programmer has to write non-erroneous processes for which we give the following typing rules.

**Definition 8.** *The typing rules for non-erroneous processes (cf. Definition 5) are*

$$(\texttt{idle})\frac{}{0 \triangleright 0} \qquad (\texttt{res})\frac{P \triangleright t}{\nu x\, P \triangleright t} \qquad \frac{P \triangleright t \qquad P' \triangleright t'}{P \mid P' \triangleright t \oplus t'}(\texttt{par}) \qquad \frac{P \triangleright t}{!P \triangleright t}(\texttt{repl})$$

$$(\texttt{inv})\frac{P \triangleright t_p \qquad I = \{\mathfrak{o}\} \times A}{s \,\&\!\leftarrow A.P \triangleright (I \cup t_p{}^{\&\leftarrow}, t_p{}^{\downarrow}, t_p{}^{?})} \qquad \frac{P \triangleright t_p \qquad Q \triangleright t_q}{\pi[\![Q]\!].P \triangleright (t_p{}^{\&\leftarrow}, t_p{}^{\downarrow}, t_q \oplus t_p{}^{?})}(\texttt{comp})$$

$$(\texttt{scope}_1)\frac{P \triangleright (I, t_c, t_u) \qquad Q \triangleright t_q}{\langle P \mid (\!|Q|\!)\rangle \triangleright ((I \cup t_c{}^{\&\leftarrow})[\![\mathfrak{o} \mapsto \mathfrak{i}]\!], t_u \oplus t_c{}^{\downarrow} \oplus t_c{}^{?} \oplus t_q, 0)} \qquad \frac{P \triangleright 0}{\langle P \mid (\!|Q|\!)\rangle \triangleright 0}(\texttt{scope}_2)$$

*where, for* $I \subset \{\mathfrak{i}, \mathfrak{o}\} \times \mathcal{A}$, $I[\![\mathfrak{o} \mapsto \mathfrak{i}]\!] \stackrel{def}{=} \{(\mathfrak{i}, a) : (\mathfrak{o}, a) \in I\} \cup (I \cap \{\mathfrak{i}\} \times \mathcal{A})$.

The first five rules are straightforward. Rule (comp) states that the type of the installation of a compensation $Q$ records the invocations in $Q$ as possible invocations of $P$ by adding them to the third component of the type of $\pi[\![Q]\!].P$. The last two rules regulate the typing of transactional scopes. By rule $(\texttt{scope}_1)$, when $P$ is in a transactional scope the invocations done by the compensations installed by $P$ (recorded in $t_u$) become possible; therefore they are removed from the third component and added to the second component with the compensations nested in $P$ (recorded in ${t_c}^{\downarrow}$ and ${t_c}^{?}$) and to those of $Q$ (recorded in $t_q$). Also, ${t_c}^{\&\leftarrow}$ records the invocation of the compensation of $P$ when $P$ is itself defined s a transactional scope (e.g., $P = \langle Q \mid (\!|C|\!)\rangle$); in this case the compensations of $P$ will be surely executed inside a transactional scope thus they are included in the first component with the substitution $[\![\mathfrak{o} \mapsto \mathfrak{i}]\!]$. A transactional scope whose process does not invoke/install anything is simply typed as 0 by rule $(\texttt{scope}_2)$.

*Example 7.* Consider the process $P = \pi_1[\![Q]\!]$ where

$$Q = \pi_2[\![R]\!] \qquad \text{and} \qquad R = s_1 \,\&\!\leftarrow A_1.\pi_3[\![s_2 \,\&\!\leftarrow A_2]\!]$$

The typing of $P$ is $t = (0, 0, (0, 0, (I_1, 0, I_2)))$ as proved by the type inference below.

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{I_2 = \{\mathfrak{o}\} \times A_2 \qquad 0 \triangleright 0}{s_2 \,\&\!\leftarrow A_2 \triangleright (I_2, 0, 0)}(\texttt{inv}) \qquad 0 \triangleright 0}{\pi_3[\![s_2 \,\&\!\leftarrow A_2]\!] \triangleright (0, 0, I_2)}(\texttt{Comp}) \qquad I_1 = \{\mathfrak{o}\} \times A_1}{s_1 \,\&\!\leftarrow A_1.\pi_3[\![s_2 \,\&\!\leftarrow A_2]\!] \triangleright (I_1, 0, I_2)}(\texttt{inv}) \qquad 0 \triangleright 0}{\pi_2[\![R]\!] \triangleright (0, 0, (I_1, 0, I_2))}(\texttt{Comp}) \qquad 0 \triangleright 0}{\pi_1[\![Q]\!] \triangleright (0, 0, (0, 0, (I_1, 0, I_2)))}(\texttt{Comp})$$

◇

**Proposition 3.** *For each non-erroneous $P \in \mathcal{P}$ there is a unique type $t$ such that $P \triangleright t$.*

**Proposition 4.** *For any non-erroneous $P, Q \in \mathcal{P}$, if $P \equiv Q$ and $P \triangleright t$ then $Q \triangleright t$.*

**Definition 9.** *Let $t$ be a type. The* flat type $\widehat{t}$ of $t$ *is defined as follows:*

$$\widehat{\mathbf{0}} = \emptyset \qquad \widehat{t} = t^{\&} \cup \textsc{Flatten}(t^{\downarrow}),\ \textit{if } t \neq \mathbf{0}$$

$$\textsc{Flatten}(\mathbf{0}) = \emptyset \qquad \textsc{Flatten}(t) = t^{\&} \cup \textsc{Flatten}(t^{\downarrow}) \cup \textsc{Flatten}(t^{?}),\ \textit{if } t \neq \mathbf{0}$$

Notice that $\widehat{t \oplus t'} = \widehat{t} \cup \widehat{t'}$. In the interpretation of $t$ as a tree, the flat type of $t$ is the union of the set labelling all the nodes of $t$, excluding those of the subtree $t^{?}$ which corresponds to dead code (cf. Example 8); in other words, either the typed process is outside a scope (in which case its pending compensations can be ignored) or the typed process is inside a scope (hence $t^{?}$ is empty because of rule `(Scope1)`).

### 4.3 Well-typedness in ATc

The definition of well-typedness requires some care. In ATc, invocations to services can be statically typed as transactional or not. However, there is a different notion of well-typedness to adopt for services.

If $P$ is not published as a service then it is possible to determine the nature of the service invocations of $P$ by inspecting its code. Therefore, it suffices to specify, for each service invocation, the attributes for which no run-time errors are possible. This enables us to adopt the following definition.

**Definition 10.** *Let $P \in \mathcal{P}$ such that $P \triangleright t$. The process $P$ is* well-typed *iff* $(\mathfrak{o}, \mathtt{m}) \notin \widehat{t}$.

*Example 8.* Process $P$ in Example 7 is (trivially) well-typed since $\widehat{t} = \emptyset$. In fact, the only service invocations of $P$ are in the compensations to install (that are dead code since $P$ is not included in any transactional scope). ◇

The correctness of a process depends also on the correctness of the services that it invokes. Remarkably, the fact that the invoked service is well-typed could be guaranteed by the service provider (as part of the service interface) and required as an obligation by the service requester in the service discovery phase. Namely, negotiation of transactional attributes should be part of the "contract" between requester and provider. The study of the mechanisms used to require/negotiate/certify transactional aspects of published services is out of the our scopes. However, we remark that our type systems provides an effective framework to certify compatibility of transactional aspects between services and invokers.

Ensuring correctness for services is a bit more complex. Whether or not the invocations in the body of (the end-point of) a service, say $s$, are transactional depends on which attribute $s$ supports and if the invocation to $s$ happened from within or outside a transactional scope. Therefore, well-typedness of services takes into account both cases.

**Definition 11.** *Let $\gamma$ be a container and $s$ be a service such that $\gamma(s) = (a, P)$ for some $a \in \mathcal{A}$ and $P \in \mathcal{P}$. Service $s$ is* well-typed in $\gamma$*, if both (3) and (4) below hold.*

$$\langle P \rangle \triangleright t \wedge a \in \{\mathtt{r}, \mathtt{rn}, \mathtt{m}, \mathtt{s}\} \implies (\mathfrak{o}, \mathtt{m}) \notin \widehat{t} \tag{2}$$

$$P \triangleright t \wedge a \in \{\mathtt{s}, \mathtt{n}, \mathtt{ns}\} \implies (\mathfrak{o}, \mathtt{m}) \notin \widehat{t} \tag{3}$$

*An environment $\Gamma$ is* well-typed *iff all the services in the domain of any $\gamma \in \Gamma$ are well-typed.*

We only consider the errors generated by the invocation of a service when attributes and transactional scopes mismatch. Errors due to other causes (e.g., failure of a communication channel) have been modelled in [5] by introducing *observers*, namely processes which can interfere in communications.

*Example 9.* Suppose that the process $P$ in Example 7 is the body of a service $s$ supporting $\mathtt{s} \in \mathcal{A}$. Both the well-typedness of $P$ and of $\langle P \rangle$ must be checked. As argued in Example 8, $P$ is well-typed while for $\langle P \rangle$ we just need to apply rule (Scope1) as follows:

$$\frac{\pi_1 [\![Q]\!] \triangleright (0, 0, (0, 0, (I_1, 0, I_2))) \qquad 0 \triangleright 0}{\langle \pi_1 [\![Q]\!] \mid (\!|0|\!) \rangle \triangleright (0, (0, 0, (I_1, 0, I_2)), 0)} \text{ (Scope1)}$$

Clearly, well-typedness of $\langle P \rangle$ depends on whether $(\mathfrak{o}, \mathfrak{m}) \in I_1 \cup I_2$ or not. $\diamond$

**Theorem 1.** *Let $P \in \mathcal{P}$ be well-typed. For every well-typed environment $\Gamma$, if $\Gamma \vdash P \rightsquigarrow \Gamma \vdash Q$ then $Q$ is well-typed.*

A straightforward corollary of Theorem 1 is

**Corollary 1.** *If $\Gamma$ and $P \in \mathcal{P}$ are well-typed and $\Gamma \vdash P \rightsquigarrow \Gamma \vdash Q$ then $Q$ is a non-erroneous process.*

Our notion of well-typedness is stricter than necessary. In fact, a weaker notion can be adopted by taking a definition of flat type where the labels of some of the 'right children' of types are not considered. Though yielding less restrictive types, this would make the theory more complex, therefore we opted for simplicity rather than generality.

## 5 ATc Type System at Work

The type system presented in § 4 checks that any possible invocation to a service requires a safe set of attributes so to avoid errors due to misuse of transactional scopes and attributes.

The design of SOC transactions could be easier if we knew, for each service invocation in a process, the maximal set of attributes that satisfies the typing. As a matter of fact, specifying a larger set of attributes in a service invocation increases the chances of finding a suitable service supporting one of the attributes. The trade-off is however that a too large set of attributes may cause a run-time error due to a service instance running in a wrongly nested transactional scopes.

Arguably, non well-typed processes can be turned into well-typed ones by changing the attributes of some invocations. We show through an example a method for designing a well-typed process based on an alternative usage of the typing system in § 4. First, consider the types obtained as in Definition 7 but for set $\mathcal{A}$ which is replaced by an infinite countable set $\Xi$ of symbolic identifiers. A *symbolic* process corresponding to $P \in \mathcal{P}$ is a term $sym(P)$ obtained by replacing each set of attributes with a distinct formal identifier in $\Xi$ meant to be substituted by a subset of $\mathcal{A}$.

*Example 10.* A symbolic process corresponding to $P_{\text{bookTheatre}}$ in Example 2 is

$$sym(P_{\text{bookTheatre}}) = \langle s_{\text{tickets}} \,\&\!\leftarrow\, X_1.\overline{\text{askSeat}}.\text{getSeat}.\overline{\text{pay}}[\![\overline{\text{getRefund}}]\!] \mid (\!|s_{\text{compensate}} \,\&\!\leftarrow\, X_2|\!)\rangle$$

(the sets of attributes in $P_{\text{bookTheatre}}$ are replaced by $X_1$ and $X_2$). ◇

A *maximal process* is a well-typed process for which augmenting any of the sets of attributes of its invocations yields the same process or a non-well typed process. Given a well-typed $P \in \mathcal{P}$, $\max(P)$ is the maximal process corresponding to $P$. (Notice that if $P$ does not make any invocation then $P = \max(P)$.)

The typing system of Definition 8 is adapted to symbolic processes by replacing rule `(inv)` with

$$\texttt{(invSym)}\ \frac{P \triangleright t_p \qquad I = \{\mathfrak{o}\} \times X}{s \,\&\!\leftarrow\, X.P \triangleright (I \cup {t_p}^{\&\!\leftarrow}, {t_p}^{\downarrow}, {t_p}^{?})}, \quad \text{where } X \text{ not occurs in } P$$

*Example 11.* By straightforward application of the typing system for symbolic processes $sym(P_{\text{bookTheatre}}) \triangleright t_{sym}$, where $t_{sym} = (\{\mathfrak{i}\} \times X_1, (\{\mathfrak{o}\} \times X_2, 0, 0), 0, 0)$. Hence, the flattened type of $sym(P_{\text{bookTheatre}})$ is $\widehat{t_{sym}} = \{\{\mathfrak{i}\} \times X_1,\ \{\mathfrak{o}\} \times X_2\}$. ◇

Finally, the maximal process is obtained by replacing each formal identifies with a suitable set of attributes. For example,

$$\max(P_{\text{bookTheatre}}) = \langle s_{\text{tickets}} \,\&\!\leftarrow\, \mathcal{A}.\overline{\text{askSeat}}.\text{getSeat}.\overline{\text{pay}}[\![\overline{\text{getRefund}}]\!] \mid (\!|s_{\text{compensate}} \,\&\!\leftarrow\, \mathcal{A} \setminus \{m\}|\!)\rangle$$

is obtained by replacing $X_1$ with $\mathcal{A}$ and $X_2$ with $\mathcal{A} \setminus \{\mathfrak{m}\}$ in $sym(P_{\text{bookTheatre}})$. In fact, the invocation to $s_{\text{tickets}}$ (i.e., the one associated with $X_1$) can possibly contain all attributes since they are transactional while the other invocation (i.e., the one to $s_{\text{compensate}}$ associated with $X_2$) can contain all attributes except $\mathfrak{m}$.

In general, one is interested only in some policies for transactional scopes and will typically choose, for each invocation in a process $P$, a subset of the attributes of the corresponding invocation in $\max(P)$.

## 6 A Testing Theory for ATc

In this section we cast the testing theory for ATc. First, we define observers for ATc in § 6.1, then we give the may and must equivalences in 6.2, and finally we show how transactional attributes change the observable behaviour of ATc systems via an example in § 6.3.

### 6.1 Observed Systems

In this section, we define on observational semantics of ATc systems (hereafter, ranged over by $S, S', \ldots$) based on testing equivalence [15]. The intuition behind testing equivalences is that an *observer* checks whether a system passes or not a test by interacting with the system. Two systems are equivalent if they pass the same tests.

**Definition 12.** *An* observer *is derived by the following grammar:*

$$\begin{array}{rll} O ::= & \mathtt{0} & \textit{empty process} \\ \mid & \checkmark & \textit{success} \\ \mid & \pi.O & \textit{prefix} \\ \mid & \lightning\pi.O & \textit{failure} \\ \mid & O + O & \textit{sum} \\ \mid & \mathbf{rec}\ X.O & \textit{recursion} \\ \mid & X & \textit{variable} \end{array}$$

*The* structural congruence for observers *is the smallest equivalence relation closed under the monoidal axioms of + and it is denoted as* $\equiv_o$.

We consider sequential observers. Failing and successful tests are represented by $\mathtt{0}$ and $\checkmark$, respectively; prefix $\pi.O$ allows observes to communicate with the system, while prefix $\lightning\pi.O$ causes the failure of $\overline{\pi}$ in the system and continues as $O$; observers can be composed with the external choice operator + and recursively defined as **rec** $X.O$ (where the occurrences of $X$ in $O$ are supposed guarded by prefixes). An observer is a process that can interact with a system over its (free) channels and trigger failures in the communications (e.g., to check that failures are correctly handled).

Notice that observers cannot be composed in parallel and therefore they do not communicate among themselves. This, and the fact that ATc does not feature name passing, allow us to avoid using name restriction in observers.

The failure of an action $\pi$ is imposed by the observer $\lightning\pi.O$. As clear from Definition 13, communication failures could possibly trigger the compensations associated to the scopes (if any) enclosing failing processes.

**Definition 13.** *The set* States *of* observed system *is the set of pairs made of a system* $S$ *and an observer* $O$ *(written as* $S \parallel O$*).*

*The reduction relation of ATc observed systems is the least relation* $\rightsquigarrow$ *satisfying the following rules:*

$$\texttt{(os-tick)}\quad S \parallel \checkmark \rightsquigarrow S \parallel \checkmark \qquad\qquad \Gamma \vdash \mathbf{C}[\pi.P] \parallel \overline{\pi}.O \rightsquigarrow \Gamma \vdash \mathbf{C}[P] \parallel O \quad \texttt{(os-comm)}$$

$$\Gamma \vdash \mathbf{C}[\langle \pi.P \mid R \mid (\!|Q|\!) \rangle] \parallel \lightning\pi.O \rightsquigarrow \Gamma \vdash \mathbf{C}[Q] \parallel O \quad \texttt{(os-fail-in)}$$

$$\Gamma \vdash \mathbf{C}[\pi.P] \parallel \lightning\pi.O \rightsquigarrow \Gamma \vdash \mathbf{C}[\mathtt{err}] \parallel O, \textit{ if } \mathbf{C}[\square]\ \textit{s-a} \qquad \texttt{(os-fail-out)}$$

$$\texttt{(os-s)}\ \frac{S \rightsquigarrow S'}{S \parallel O \rightsquigarrow S' \parallel O} \qquad\qquad \frac{\Gamma \vdash P \parallel O \rightsquigarrow \Gamma \vdash P' \parallel O'}{\Gamma \vdash P \parallel O + O'' \rightsquigarrow \Gamma \vdash P' \parallel O'}\ \texttt{(os-sum)}$$

$$\frac{O \equiv_o O_1 \quad S \parallel O_1 \rightsquigarrow S' \parallel O_2 \quad O_2 \equiv_o O'}{S \parallel O \rightsquigarrow S' \parallel O'}\ \texttt{(os-cong)}$$

Axiom `(os-tick)` signals that when a test is passed. Axiom `(os-comm)` models a communication step involving (a part of) the system and the observer. Axiom `(os-fail-in)` describe triggers the compensation when there is a communication failure within a transactional scope. Axiom `(os-fail-out)` yields an error when a failure outside a transactional scope. Rule `(os-s)` models a step due to transitions of

the system that do not involve the observer. The interactions of the system with non-deterministic observers are defined by rule (os-sum); notice that, by (os-tick), if $O = \checkmark$, the other branch $O''$ is discarded[6]. Rule (os-cong) is the usual rule for congruence.

A computation is a (possibly infinite) sequence of states $S_0 \parallel O_0, \ldots, S_n \parallel O_n, \ldots$ such that $S_i \parallel O_i \rightsquigarrow S_{i+1} \parallel O_{i+1}$ for each $i$. We denote with Comp (ranged over by $c$) the set of all the computations.

### 6.2 Testing Equivalences for ATc

Two basic elements of the testing theory are the notions of *successful* and *non-divergent* computation. Intuitively, a computation is successful if the test is passed (i.e., the corresponding observer halts with $\checkmark$). Non-divergent computations are successful computations that reach $\checkmark$ before the occurrence of an error. We now cast the basic notions of the testing theory to ATc observed systems.

**Definition 14.** *Let $O \searrow \checkmark$ stand for $O = \checkmark + O'$ for some observer $O'$. A state $\Gamma \vdash P \parallel O$ is successful if $O \searrow \checkmark$ (we let Success $= \{S \parallel O \in \mathsf{States} : O \searrow \checkmark\}$).*

- *$\Gamma \vdash P \parallel O \in \mathsf{States}$ is* diverging *if $P = \mathbf{C}[\mathtt{err}]$ for a context $\mathbf{C}[\natural]$;*
- *a computation $c \in \mathsf{Comp}$ is* successful *if it contains $S \parallel O \in \mathsf{Success}$ and* unsuccessful *otherwise;*
- *a computation $c = S_0 \parallel O_0, S_2 \parallel O_2, \ldots, S_n \parallel O_n, \ldots$* diverges *if either*
  1. *$c$ is unsuccessful or*
  2. *there is $i \geq 0$ such that $S_i \parallel O_i$ is diverging and $O_j \not\searrow \checkmark$ for $j < i$.*

*Hereafter, we write $S \parallel O \searrow \mathtt{err}$ when $S \parallel O$ is a diverging state, $c \Uparrow$ if $c$ is an unsuccessful computation, and $c \Downarrow$ if not $c \Uparrow$.*

As customary for the testing theory, we use $\bot$ and $\top$ to denote divergence and non divergence, respectively. The possible outcomes of a computation are defined in terms of *result sets*, namely (non-empty) subsets of $\{\top, \bot\}$.

**Definition 15.** *The* result set of $S \parallel O \in \mathsf{States}$, $\mathfrak{R}(S \parallel O) \subseteq \{\top, \bot\}$, *is defined by*

- *$\top \in \mathfrak{R}(S \parallel O)$ iff there is a successful $c \in \mathsf{Comp}$ that starts from $S \parallel O$,*
- *$\bot \in \mathfrak{R}(S \parallel O)$ iff there is $c \in \mathsf{Comp}$ starting from $S \parallel O$ such that $c \Uparrow$.*

As in [15], we consider *may* and *must* preorders and the corresponding induced equivalences. Recall that (*i*) may-testing enforces some *fairness* which ensures that there divergence is not "catastrophic" provided that there is a chance of successful and (*ii*) that must-testing corresponds to liveness as it requires all possible computation to be successful.

**Definition 16.** *Given a system $S$ and an observer $O$, we say that*

$$S \text{ MAY } O \iff \top \in \mathfrak{R}(S \parallel O) \qquad \text{and} \qquad S \text{ MUST } O \iff \{\top\} = \mathfrak{R}(S \parallel O)$$

*We define the preorders $\sqsubseteq_{\mathbf{m}}$ (may preorder) and $\sqsubseteq_{\mathbf{M}}$ (must preorder) on systems:*

[6] Also the $\checkmark$ branch can be discarded according to rule (os-sum). This is necessary to define the testing equivalences (cf. § 6.2).

– $S \sqsubseteq_{\mathbf{m}} S' \iff (S \text{ MAY } O \implies S' \text{ MAY } O)$, *for all observers*
– $S \sqsubseteq_{\mathbf{M}} S' \iff (S \text{ MUST } O \implies S' \text{ MUST } O)$, *for all observers.*

*The two equivalences* $\simeq_{\mathbf{m}}$ *and* $\simeq_{\mathbf{M}}$ *corresponding to* $\sqsubseteq_{\mathbf{m}}$ *and* $\sqsubseteq_{\mathbf{M}}$ *are defined as expected:* $\simeq_{\mathbf{m}} = \sqsubseteq_{\mathbf{m}} \cap \sqsubseteq_{\mathbf{m}}^{-1}$ *and* $\simeq_{\mathbf{M}} = \sqsubseteq_{\mathbf{M}} \cap \sqsubseteq_{\mathbf{M}}^{-1}$.

### 6.3 An Example on Observed Behaviour and Attributes

Consider a scenario where a service $s$ acts as a proxy of a shared resource (not explicitly represented); the body $R$ of $s$ is

$$R = \overline{\mathrm{a}}[\![\overline{\mathrm{r}}]\!].\mathbf{rec}\ X.(\mathrm{u}.X + \mathrm{q}.\overline{\mathrm{r}})$$

The expected behaviour is that, upon invocation, $s$ interact with the resource and immediately locks it ($\overline{\mathrm{a}}$); then it grants the use of the resource to the client as long as the latter requests it (u); when the client ends the computations (q), the resource is released (r). Observe that, if any error occurs after the invocation, the compensation of $\overline{\mathrm{a}}$ (namely, $\overline{\mathrm{r}}$) will (correctly) release the resource.

Assume that $\Gamma$ is an environment such that $R \in \Gamma(\mathtt{r}, s)$ and $R \in \Gamma(\mathtt{rn}, s)$, namely there are (at least) two providers for $s$ with the same body $R$ but supporting different attributes.

Consider the two possible (well-typed) clients $P_1$ and $P_2$ below (their compensations are immaterial hence omitted)

$$P_1 = \langle s \,\&\!\!\leftarrow \{\mathtt{r}\}.\overline{\mathrm{u}}.\overline{\mathrm{q}} \mid (\!|\cdots|\!) \rangle \qquad\qquad P_2 = \langle s \,\&\!\!\leftarrow \{\mathtt{rn}\}.\overline{\mathrm{u}}.\overline{\mathrm{q}} \mid (\!|\cdots|\!) \rangle$$

where $P_2$ is obtained by replacing the attribute $\mathtt{r}$ with $\mathtt{rn}$ in the invocation to $s$ of $P_1$. Both clients invoke $s$, use ($\overline{\mathrm{u}}$), and finally release the resource ($\overline{\mathrm{q}}$).

The different required attributes generate two behaviours. In fact, the invocations of $s$ from $P_1$ and $P_2$ result in the following systems (Definition 6, rules `(tx1)` and `(tx4)`, respectively)

$$S_1 = \Gamma \vdash \langle \overline{\mathrm{u}}.\overline{\mathrm{q}} \mid \overline{\mathrm{a}}[\![\overline{\mathrm{r}}]\!].\mathrm{u}.\mathrm{q}.\overline{\mathrm{r}} \mid (\!|\cdots|\!) \rangle$$
$$S_2 = \Gamma \vdash \langle \overline{\mathrm{u}}.\overline{\mathrm{q}} \mid (\!|\cdots|\!) \rangle \mid \langle \overline{\mathrm{a}}[\![\overline{\mathrm{r}}]\!].\mathrm{u}.\mathrm{q}.\overline{\mathrm{r}} \mid (\!|\mathsf{0}|\!) \rangle$$

Take the following observer

$$O = \mathrm{a}.(\mathrm{r}.\checkmark + \lightning\overline{\mathrm{u}}.\mathrm{r}.\checkmark)$$

that checks, after the resource is acquired, if it is released regardless possible failures on clients' requests of use ($\lightning\overline{\mathrm{u}}$). We have

$$S_1 \parallel O \rightsquigarrow \langle \overline{\mathrm{u}}.\overline{\mathrm{q}} \mid \mathrm{u}.\mathrm{q}.\overline{\mathrm{r}} \mid (\!|\cdots \mid \overline{\mathrm{r}}|\!) \rangle \parallel \mathrm{r}.\checkmark + \lightning\overline{\mathrm{u}}.\mathrm{r}.\checkmark \tag{4}$$

$$S_2 \parallel O \rightsquigarrow \langle \overline{\mathrm{u}}.\overline{\mathrm{q}} \mid (\!|\cdots|\!) \rangle \mid \langle \mathrm{u}.\mathrm{q}.\overline{\mathrm{r}} \mid (\!|\overline{\mathrm{r}}|\!) \rangle \parallel \mathrm{r}.\checkmark + \lightning\overline{\mathrm{u}}.\mathrm{r}.\checkmark \tag{5}$$

The result set of the state reached in (5) is $\{\top\}$ since the compensation $\overline{\mathrm{r}}$ will be executed in both the branches of the observer and the resource eventually released. Instead, the result set of the state reached in (6) is $\{\bot, \top\}$ since the resource is not released if the failing branch of the observer is chosen (the system is deadlocked and the resource is never released); on the contrary the resource is released if no failure arise.

In general, different attributes associated to a service call generate a different observational behaviour. However, in § 2 we show that, in some cases, it is possible to interchange the transactional attributes while preserving the same observed behaviour.

## 7 Results on Testing Equivalence for ATc

We apply the theory of testing of ATc to define a relationship between transactional attributes. Specifically, we show that, under suitable conditions, attribues are interchangeable. Namely, using an attribute instead of another in a service invocation does not alter the observational behaviour of systems. Remarkably, the observational theory presented in § 6 is used here to study some relations on transactional attributes derived from the testing preorders. Namely, we investigate some preorder relations on attributes defined in terms of testing preorders and show that systems where equivalent attribute are replaced with each other have the same observable behaviour.

*Example 12.* The example in § 6.3 illustrates how the choice of different attributes yields different observable behaviours. Consider again the process $P_1 = \langle s \,\&\!\leftarrow\, \{\mathtt{r}\}.\overline{\mathtt{u}}.\overline{\mathtt{q}} \mid (\!|\cdots|\!)\rangle$ of § 6.3. Since the invocation to $s$ is transactional, then the observational behaviour of $P_1$ does not change by replacing $\mathtt{r}$ with $\mathtt{m}$ or $\mathtt{s}$. ◇

For $\mu \in \{\mathbf{m}, \mathbf{M}\}$, we define the preorders $\leq_\mu \subseteq \mathcal{A} \times \mathcal{A}$, ($\mathbf{m}$ stands for *may* while, $\mathbf{M}$ stands for *must*). Such preorders correspond to the preorders $\sqsubseteq_{\mathbf{m}}$ and $\sqsubseteq_{\mathbf{M}}$ on systems. The intuition is that if $S'$ is obtained by replacing $a_2$ for $a_1$ in a system $S$ and $a_1 \leq_\mu a_2$, then $S \sqsubseteq_\mu S'$. Preorder $\leq_\mu$ is defined in terms of preorders $\leq^{\mathrm{o}}_\mu$ and $\leq^{\mathrm{i}}_\mu$ that differ on how they act on how they subsitute attributes in transactional scopes.

Before giving $\leq_\mu$ (cf. Definition 17), two substitutions $[^{-}/_{\_}]^{\mathrm{i}}$ and $[^{-}/_{\_}]^{\mathrm{o}}$ have to be defined. These substitutions respectively replace the attributes that appear in transactional invocations or outside transactional scope. The complete definitions are given in Appendix B (Definition 7) according to the standard attribute substitution also given in Appendix B; here we consider only the most important cases.

Substitution $[^{-}/_{\_}]^{\mathrm{o}}$ acts as the standard substitution except from the case of transactional scope; in fact

$$\langle P \mid (\!|Q|\!)\rangle[^{b}/_{a}]^{\mathrm{o}} = \langle P \mid (\!|(Q[^{b}/_{a}]^{\mathrm{o}})|\!)\rangle$$

namely, the running process $P$ is not subject to further substitutions, while the compensation $Q$ is (since it would be executed outside the current transactional scope).

Substitution $[^{-}/_{\_}]^{\mathrm{i}}$ acts as expected but for the service invocation and transactional scope cases, respectively defined as

$$(s \,\&\!\leftarrow\, A.P)[^{b}/_{a}]^{\mathrm{i}} = s \,\&\!\leftarrow\, A.(P[^{b}/_{a}]^{\mathrm{i}}) \tag{6}$$

$$\langle P \mid (\!|Q|\!)\rangle[^{b}/_{a}]^{\mathrm{i}} = \langle P[a/b] \mid (\!|Q[^{b}/_{a}]^{\mathrm{i}}|\!)\rangle \tag{7}$$

In (7), the substitution does not change the attributes of the invocation to $s$, since the process is outside a transactional scope. In (8), the standard substitution of $a$ for $b$ is applied to the running process of the scope $P$ (the substitutions has to act on all the invocations in $P$) while $[^{b}/_{a}]^{\mathrm{i}}$ is applied to the compensation $Q$ (as $Q$ could be executed outside a transactional scope).

A standard attribute substitution $\sigma$ is applied to a system $\Gamma \vdash P$ by setting

$$(\Gamma \vdash P)\sigma = \Gamma\sigma \vdash P\sigma \quad \text{where} \quad \Gamma\sigma = \begin{cases} \emptyset, & \Gamma = \emptyset \\ \Gamma'\sigma \cup \{\gamma, upd(\gamma, \sigma)\}, & \Gamma = \Gamma' \cup \{\gamma\} \end{cases}$$

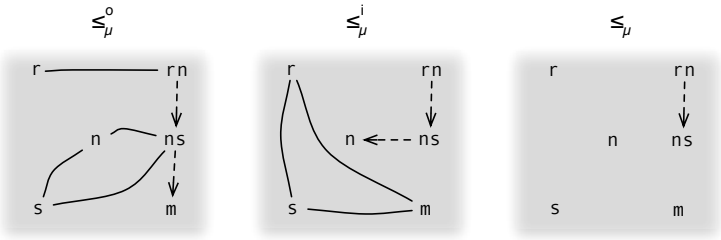



**Fig. 3.** Order on attributes

and $upd(\gamma, \sigma) : s \mapsto (\sigma(a), R))$ for each $s$ such that $\gamma(s) = (a, R)$. Also, $[\text{-}/\_]^{\mathrm{i}}$ are extended on systems by letting $(\Gamma \vdash P)[^{b}/_{a}]^{\mathrm{i}} = \Gamma[^{b}/_{a}] \vdash (P[^{b}/_{a}]^{\mathrm{i}})$ and similarly for $[\text{-}/\_]^{\mathrm{o}}$. Finally, $(S \parallel O)\sigma = S\sigma \parallel O$ for any attribute substitution $\sigma$, namely substitutions do not affect observers.

The relationships between transactional attributes are formally defined as follows:

**Definition 17.** *Let $a_1, a_2 \in \mathcal{A}$ and $\mu \in \{\mathbf{m}, \mathbf{M}\}$,*

- $a_1 \leq^{\mathrm{o}}_{\mu} a_2 \iff S \sqsubseteq_{\mu} S[^{a_1}/_{a_2}]^{\mathrm{o}}$, *for any system $S$*
- $a_1 \leq^{\mathrm{i}}_{\mu} a_2 \iff S \sqsubseteq_{\mu} S[^{a_1}/_{a_2}]^{\mathrm{i}}$, *for any system $S$.*

*We let $\leq_{\mu} \;=\; \leq^{\mathrm{i}}_{\mu} \cap \leq^{\mathrm{o}}_{\mu}$ and $=_{\mu} \;=\; \leq_{\mu} \cap \leq_{\mu}$.*

Figure 3 illustrates the relationships between attributes associated to non transactional invocations (leftmost box), transactional invocations (middle box), and both transactional and non transactional invocations (rightmost box). The bidirectional relationships are represented by arrows without edges. Solid lines are the equivalences proved in Theorem 2 while the preorder relations represented by the dashed arrows are conjectured to hold for subclasses of systems that can be suitably characterised using the types in § 4.

**Lemma 1.** *If $a_1, a_2 \in \{\mathtt{r}, \mathtt{rn}\}$, for any system $S$*

$$S \rightsquigarrow S' \implies S[^{a_2}/_{a_1}]^{\mathrm{o}} \rightsquigarrow S'[^{a_2}/_{a_1}]^{\mathrm{o}} \tag{8}$$

*Proof.* The proof is by induction on the derivation of $S \rightsquigarrow S'$ (cf. Definition 6); we let $\sigma^{\mathrm{o}} = [^{a_2}/_{a_1}]^{\mathrm{o}}$ and $\sigma = [^{a_2}/_{a_1}]$, and given $A \subseteq \mathcal{A}$, $A\sigma^{\mathrm{o}}$ stand for $A \setminus \{a_1\} \cup \{a_2\}$.

Firstly, consider the axioms `(ntx1÷3)` for which $S = \Gamma \vdash \mathbf{C}[\square] \circ s \,\&\!\!\leftarrow\, A.P$ for a s-a context $\mathbf{C}[\square]$. By definition, $S\sigma^{\mathrm{o}} = \Gamma\sigma \vdash \mathbf{C}[\square]\sigma^{\mathrm{o}} \circ s \,\&\!\!\leftarrow\, A\sigma^{\mathrm{o}}.P\sigma^{\mathrm{o}}$ and

- if $S \rightsquigarrow S'$ is obtained by axiom `(ntx1)` then $S' = \Gamma \vdash \mathbf{C}[\mathtt{err}]$ and $\mathtt{m} \in A\sigma^{\mathrm{o}}$, therefore $S\sigma^{\mathrm{o}} \rightsquigarrow \Gamma\sigma \vdash \mathbf{C}[\square]\sigma^{\mathrm{o}} \circ \mathtt{err}\sigma^{\mathrm{o}}$ (by `(ntx1)`) and $\Gamma\sigma \vdash \mathbf{C}[\square]\sigma^{\mathrm{o}} \circ \mathtt{err}\sigma^{\mathrm{o}} = \Gamma\sigma \vdash \mathbf{C}[\mathtt{err}]\sigma^{\mathrm{o}} = S'\sigma^{\mathrm{o}}$ (by Proposition 6 in Appendix B);
- if $S \rightsquigarrow S'$ is obtained by axiom `(ntx3)` then $S' = \Gamma \vdash \mathbf{C}[P] \mid \langle R \rangle$ with $R \in \Gamma(s, \{\mathtt{r}, \mathtt{rn}\} \cap A)$. Then $\Gamma\sigma(s) = (\sigma^{\mathrm{o}}(a), R\sigma^{\mathrm{o}})$ with $a = a_1 \vee a = a_2$, hence $R\sigma^{\mathrm{o}} \in \Gamma\sigma(s, \{a\} \cap A\sigma^{\mathrm{o}})$. Therefore $S\sigma^{\mathrm{o}} \rightsquigarrow \Gamma\sigma \vdash \mathbf{C}[\square]\sigma^{\mathrm{o}} \circ (\mathbf{C}[P] \mid \langle R \rangle)\sigma^{\mathrm{o}}$ (by `(ntx3)`) and $\Gamma\sigma \vdash \mathbf{C}[\square]\sigma^{\mathrm{o}} \circ (\Gamma \vdash \mathbf{C}[P] \mid \langle R \rangle)\sigma^{\mathrm{o}} = \Gamma\sigma \vdash \mathbf{C}[P]\sigma^{\mathrm{o}} \mid \langle R\sigma^{\mathrm{o}} \rangle = S'\sigma^{\mathrm{o}}$ (by Proposition 6 in Appendix B).

The proof in the case where $S \rightsquigarrow S'$ is obtained by axiom (ntx2) is similar to the previous case.

Secondly, consider (tx1) where $S = \Gamma \vdash \mathbf{C}[\square] \circ \langle s \,\&\!\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q|\!)\rangle$ for a context $\mathbf{C}[\square]$ and $S' = \Gamma \vdash \mathbf{C}[\square] \circ \langle P_1 \mid P_2 \mid (\!|Q|\!)\rangle$. Since $R \in \Gamma(s, A)$ then, by definition of $\Gamma\sigma$, $R \in \Gamma(s, (A \cap \{\mathtt{m}, s, r\})\sigma^{\mathrm{o}})$. We have the following two cases:

- If $\mathbf{C}[\square]$ is not s-a, then there exists a context $\mathbf{C}''[\square]$ such that $\mathbf{C}[\square] = \mathbf{C}''[\square] \circ \langle \square \mid R \mid (\!|Q''|\!)\rangle$. Hence, $S\sigma^{\mathrm{o}} = \Gamma\sigma \vdash \mathbf{C}''[\square]\sigma^{\mathrm{o}} \circ \langle\langle s \,\&\!\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q|\!)\rangle \mid R \mid (\!|Q''|\!)\rangle\sigma^{\mathrm{o}} = \Gamma\sigma \vdash \mathbf{C}''[\square]\sigma^{\mathrm{o}} \circ \langle\langle s \,\&\!\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q|\!)\rangle \mid R \mid (\!|Q''\sigma^{\mathrm{o}}|\!)\rangle$ = (i.e., the substitution does not involve the process inside the inner transactional scope nor its compensation). Therefore, the thesis follows by (tx1).
- If $\mathbf{C}[\square]$ is s-a, then $S\sigma^{\mathrm{o}} = \Gamma\sigma \vdash \mathbf{C}[\square]\sigma^{\mathrm{o}} \circ \langle s \,\&\!\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q|\!)\rangle\sigma^{\mathrm{o}} = \Gamma\sigma \vdash \mathbf{C}[\square]\sigma^{\mathrm{o}} \circ \langle s \,\&\!\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q\sigma^{\mathrm{o}}|\!)\rangle$ (i.e., the substitution does not involve the process inside the transactional scope but it involves its compensation). In this case $S\sigma^{\mathrm{o}} \rightsquigarrow \Gamma\sigma \vdash \mathbf{C}[\square]\sigma^{\mathrm{o}} \circ \langle P_1 \mid P_2 \mid (\!|Q\sigma^{\mathrm{o}}|\!)\rangle$ (by (tx1)) and $\Gamma\sigma \vdash \mathbf{C}[\square]\sigma^{\mathrm{o}} \circ \langle P_1 \mid P_2 \mid (\!|Q\sigma^{\mathrm{o}}|\!)\rangle = S'\sigma^{\mathrm{o}}$

The proof in the case where $S \rightsquigarrow S'$ is obtained by axiom (tx2÷4) is similar.

Finally, consider the axiom (s-p) for which $S = \Gamma \vdash P$ and by definition, $S\sigma^{\mathrm{o}} = \Gamma\sigma \vdash P\sigma^{\mathrm{o}}$. If $S \rightsquigarrow S'$, where $S' = \Gamma \vdash P'$ then, since all the possible process moves $P \to P'$ do not involve the transactional attributes then also $P\sigma^{\mathrm{o}} \to P''$. It is straightforward to observe that $P'' = P'\sigma^{\mathrm{o}}$:

- If $P \to P'$ by axiom (p1) then $P = \mathbf{C}[\langle \pi[\![Q]\!]_1.P_1 \mid (\!|R_1|\!)\rangle] \mid \mathbf{C}'[\langle \bar{\pi}[\![Q_2]\!].P_2 \mid (\!|R_2|\!)\rangle]$ and $P' = \mathbf{C}[\langle P_1 \mid (\!|R_1 \mid Q_1|\!)\rangle] \mid \mathbf{C}'[P_2]$.
  - If $\mathbf{C}[\square]$ is not s-a then there exists context $\mathbf{C}''[\square]$ such that $\mathbf{C}[\square] = \mathbf{C}''[\square] \circ \langle \square \mid R \mid (\!|Q''|\!)\rangle$. $P\sigma^{\mathrm{o}} = \mathbf{C}''[\square]\sigma^{\mathrm{o}} \circ \langle\langle \pi[\![Q]\!]_1.P_1 \mid (\!|R_1|\!)\rangle \mid R \mid (\!|Q''|\!)\rangle\sigma^{\mathrm{o}} \mid \mathbf{C}'[\langle \bar{\pi}[\![Q_2]\!].P_2 \mid (\!|R_2|\!)\rangle]\sigma^{\mathrm{o}}$, hence only $R_2$ is involved in the substitution provided that $\mathbf{C}'[\square]$ is not s-a). Therefore, $P\sigma^{\mathrm{o}} \to \mathbf{C}''[\square] \circ \langle\langle P_1 \mid (\!|R_1 \mid Q_1|\!)\rangle \mid R \mid (\!|Q''|\!)\rangle\sigma^{\mathrm{o}} = P'\sigma^{\mathrm{o}}$.
  - If $\mathbf{C}[\square]$ is s-a then $P\sigma^{\mathrm{o}} = \mathbf{C}[\square]\sigma^{\mathrm{o}} \circ \langle \pi[\![Q]\!]_1.P_1 \mid (\!|R_1\sigma^{\mathrm{o}}|\!)\rangle \mid \mathbf{C}'[\langle \bar{\pi}[\![Q_2]\!].P_2 \mid (\!|R_2|\!)\rangle]\sigma^{\mathrm{o}}$ and only $R_2$ is involved in the substitution provided that $\mathbf{C}'[\square]$ is s-a. Therefore, $P\sigma^{\mathrm{o}} \to \mathbf{C}[\square]\sigma^{\mathrm{o}} \circ \langle P_1 \mid (\!|R_1 \mid Q_1\sigma^{\mathrm{o}}|\!)\rangle = P'\sigma^{\mathrm{o}}$.

The proofs for the remaining axioms are similar while the proof for the inference rules is a straightforwad applycation of the inductive hypothesis. □

Observe that also the converse of (9) holds.

**Lemma 2.** *For any observed system $S \parallel O$*

$$S \parallel O \rightsquigarrow S' \parallel O' \implies (S \parallel O[^b/_a]^{\mathrm{o}}) \rightsquigarrow (S' \parallel O')[^b/_a]^{\mathrm{o}} \tag{9}$$

*where $a, b \in \{\mathtt{r}, \mathtt{rn}\}$.*

*Proof.* The proof is by induction on the derivation of the transition in the hypothesis of (10), hereafter referred to as $t$. Recall that, for any observed system $S \parallel O$ and attribute substitution $\sigma$, $(S \parallel O)\sigma = S\sigma \parallel O$ by definition.

First we consider the axioms of Definition 13. The proof is trivial if $t$ is an instance of `(os-tick)`. For the remaining axioms, let $S = \Gamma \vdash \mathbf{C}[\square] \circ \pi.P$ for an environment $\Gamma$, a context $\mathbf{C}[\square]$, and $\pi.P \in \mathcal{P}$.

- If $t$ holds by axiom `(os-comm)`, then $(S \parallel O)[^b/_a]^{\mathrm{o}} \rightsquigarrow \Gamma[^b/_a]^{\mathrm{o}} \vdash \mathbf{C}[\square][^b/_a]^{\mathrm{o}} \circ P[^b/_a]^{\mathrm{o}} \parallel O'$ by `(os-comm)` because $(S \parallel O)[^b/_a]^{\mathrm{o}} = \Gamma[^b/_a]^{\mathrm{o}} \vdash \mathbf{C}[\square][^b/_a]^{\mathrm{o}} \circ \pi.P[^b/_a]^{\mathrm{o}} \parallel O$ (by Proposition 6 in Appendix B).
- If $t$ holds by axiom `(os-fail-in)` then, for a context $\mathbf{C}'[\square]$, $\mathbf{C}[\square] = \mathbf{C}'[\square] \circ \langle \square \mid R \mid (\!|Q|\!) \rangle$ and the proof is similar to the previous case.
- The case `(os-fail-out)` is slightly more involved. By Proposition 6 (cf. Appendix B), $(S \parallel O)[^b/_a]^{\mathrm{o}} = \Gamma[^b/_a]^{\mathrm{o}} \vdash \mathbf{C}[\square][^b/_a]^{\mathrm{o}} \circ \pi.P[^b/_a]^{\mathrm{o}} \parallel O$ since $\mathbf{C}[\square]$ is s-a by the side condition of `(os-fail-out)`. Therefore, `(os-fail-out)` may also be applied to $(S \parallel O)[^b/_a]^{\mathrm{o}}$ obtaining $\Gamma[^b/_a]^{\mathrm{o}} \vdash \mathbf{C}[\square][^b/_a]^{\mathrm{o}} \circ \mathtt{err} \parallel O = (\mathbf{C}[\mathtt{err}] \parallel O)[^b/_a]^{\mathrm{o}}$.

We now consider the inference rules in Definition 13.

- Let $t$ be obtained with a proof ending with an application of `(os-s)`. By Lemma 1 $S[^b/_a]^{\mathrm{o}} \rightsquigarrow S'[^b/_a]^{\mathrm{o}}$ and therefore $(S \parallel O)[^b/_a]^{\mathrm{o}} \rightsquigarrow S'[^b/_a]^{\mathrm{o}} \parallel O'$ by rule `(os-s)`.
- If $t$ is obtained with a proof ending with an application of `(os-sum)`, then $S = \Gamma \vdash P$ for a $P \in \mathcal{P}$. By inductive hypothesis, $(S \parallel O)[^b/_a]^{\mathrm{o}} \rightsquigarrow (\Gamma \vdash P' \parallel O')[^b/_a]^{\mathrm{o}}$ for a $P' \in \mathcal{P}$. Hence, by rule `(os-sum)`, $(S \parallel O + O'')[^b/_a]^{\mathrm{o}} = S[^b/_a]^{\mathrm{o}} \parallel O + O'' \rightsquigarrow S'[^b/_a]^{\mathrm{o}} \parallel O' = (S' \parallel O')[^b/_a]^{\mathrm{o}}$

The case of `(os-cong)` is trivially obtained by induction. □

**Theorem 2.** *The following equivalences on $\mathcal{A}$ hold*

(a) $\mathtt{r} =^{\mathrm{o}}_{\mu} \mathtt{rn}$ (b) $\mathtt{ns} =^{\mathrm{o}}_{\mu} \mathtt{n} =^{\mathrm{o}}_{\mu} \mathtt{s}$ (c) $\mathtt{m} =^{\mathrm{i}}_{\mu} \mathtt{r} =^{\mathrm{i}}_{\mu} \mathtt{s}$

*Proof.* We prove (a) and we omit the proofs for (b) and (c) as they are similar to (a) once a lemma analogous to Lemma 2 is proved.

If $\mu = \mathbf{m}$ then and there is a successful computation $c$ from $S \parallel O$, then there is a successful computation from $(S \parallel O)[^{a_2}/_{a_1}]^{\mathrm{o}}$ (by induction of the sequence $c$ and Lemma 2.

The case $\mu = \mathbf{M}$ is proved by a contrapositive argument. If there is a non successful computation from $(S \parallel O)[^{a_2}/_{a_1}]^{\mathrm{o}}$. Then, by Lemmas 5 (cf. Appendix B) and Lemma 2, a non successful computation may be found for $S \parallel O$. □

### 7.1 Relating Different Scope-Scenarios

In general, the behaviour of a system changes depending on how its processes are nested in transactional contexts (Proposition 5). However, we illustrate how to compare systems in different transactional contexts when restricting to a non-trivial class of systems (Proposition 6).

**Proposition 5.** *For all environments $\Gamma$, there exist $P, R, Q \in \mathcal{P}$ such that:*

1. $\Gamma \vdash \langle P \mid R \mid (\!|Q|\!) \rangle \not\sqsubseteq_{\mathbf{M}} \Gamma \vdash \langle P \mid (\!|Q|\!) \rangle \mid \langle R \rangle$

2. $\Gamma \vdash \langle P \mid (\!|Q|\!) \rangle \mid \langle R \rangle \not\sqsubseteq_{\mathbf{M}} \Gamma \vdash \langle P \mid R \mid (\!|Q|\!) \rangle$
3. $\Gamma \vdash \langle P \mid R \mid (\!|Q|\!) \rangle \not\sqsubseteq_{\mathbf{M}} \Gamma \vdash \langle P \mid (\!|Q|\!) \rangle \mid \langle R \mid (\!|Q|\!) \rangle$
4. $\Gamma \vdash \langle P \mid (\!|Q|\!) \rangle \mid \langle R \mid (\!|Q|\!) \rangle \not\sqsubseteq_{\mathbf{M}} \Gamma \vdash \langle P \mid R \mid (\!|Q|\!) \rangle$
5. $\Gamma \vdash \langle P \mid R \mid (\!|Q|\!) \rangle \not\sqsubseteq_{\mathbf{M}} \Gamma \vdash \langle P \mid (\!|Q|\!) \rangle \mid R$
6. $\Gamma \vdash \langle P \mid (\!|Q|\!) \rangle \mid R \not\sqsubseteq_{\mathbf{M}} \Gamma \vdash \langle P \mid R \mid (\!|Q|\!) \rangle$

*Proof.* The proof is straightforward and consists of the following counterexamples:

1. $\Gamma \vdash \langle \overline{a} \mid \overline{b} \mid (\!|\overline{c}|\!) \rangle$ and $\Gamma \vdash \langle \overline{a} \mid (\!|\overline{c}|\!) \rangle \mid \langle \overline{b} \rangle$ are distinguished by $\lightning b.c.\checkmark$
2. $\Gamma \vdash \langle \overline{a} \mid (\!|\overline{c}|\!) \rangle \mid \langle \overline{b} \rangle$ and $\Gamma \vdash \langle \overline{a} \mid \overline{b} \mid (\!|\overline{c}|\!) \rangle$ are distinguished by $\lightning a.b.\checkmark$
3. $\Gamma \vdash \langle \overline{a}[\![\overline{e}]\!].\overline{d} \mid d.\overline{b} \mid (\!|\overline{c}|\!) \rangle$ and $\Gamma \vdash \langle \overline{a}[\![\overline{e}]\!].\overline{d} \mid (\!|\overline{c}|\!) \rangle \mid \langle d.\overline{b} \mid (\!|\overline{c}|\!) \rangle$ are distinguished by $\lightning b.e.\checkmark$
4. $\Gamma \vdash \langle \overline{a} \mid (\!|\overline{c}|\!) \rangle \mid \langle \overline{b} \mid (\!|\overline{c}|\!) \rangle$ and $\Gamma \vdash \langle \overline{a} \mid \overline{b} \mid (\!|\overline{c}|\!) \rangle$ are distinguished by $\lightning a.b.\checkmark$
5. $\Gamma \vdash \langle \overline{a} \mid \overline{b} \mid (\!|\overline{c}|\!) \rangle$ and $\Gamma \vdash \langle \overline{a} \mid (\!|\overline{c}|\!) \rangle \mid \overline{b}$ are distinguished by $\lightning b.c.\checkmark$
6. $\Gamma \vdash \langle \overline{a} \mid (\!|\overline{c}|\!) \rangle \mid \overline{b}$ and $\Gamma \vdash \langle \overline{a} \mid \overline{b} \mid (\!|\overline{c}|\!) \rangle$ are distinguished by $\lightning a.b.\checkmark$.

It is straightforward to see that in each case the observer is successful for the first process and fails on the second one. □

**Lemma 3.** *Let $P \triangleright t$ and $(\mathfrak{o}, a) \notin \widehat{t}$ (resp. $(\mathfrak{i}, a) \notin \widehat{t}$):*

- *if $P \to P'$ and $P' \triangleright t'$ then $(\mathfrak{o}, a) \notin \widehat{t'}$ (resp. $(\mathfrak{i}, a) \notin \widehat{t'}$).*
- *let $\Gamma$ be an environment whose codomain only includes processes $P_S$ such that $P_S \triangleright t_s$ and $(\mathfrak{i}, a) \notin \widehat{t_s}$ (resp. $(\mathfrak{o}, a) \notin \widehat{t_s}$) then for all observer $O$.*
  - *If $\Gamma \vdash P \rightsquigarrow \Gamma \vdash P'$ and $P' \triangleright t'$ then $(\mathfrak{o}, a) \notin \widehat{t'}$ (resp. $(\mathfrak{i}, a) \notin \widehat{t'}$).*
  - *If $\Gamma \vdash P \parallel O \rightsquigarrow \Gamma \vdash P' \parallel O'$ and $P' \triangleright t'$ then $(\mathfrak{o}, a) \notin \widehat{t'}$ (resp. $(\mathfrak{i}, a) \notin \widehat{t'}$).*

If attributes are used "wisely", then systems may preserve their behaviour in different transactional contexts. Interestingly, a characterisation of such class of systems can be given in terms of our types.

**Definition 18.** *A process $P \in \mathcal{P}$ is* prudent *if $P \triangleright t$ and $\widehat{t} \cap \{(\mathfrak{o}, \mathtt{r}), (\mathfrak{i}, \mathtt{n})\} = \emptyset$. A system $\Gamma \vdash P$ is* prudent *if $P$ and all the processes in the codomain of $\Gamma$ are prudent.*

Intuitively, a system is prudent when it does not deliberately put unmatchable requirements on its invocations.

**Proposition 6.** *If $\Gamma \vdash \langle P \mid (\!|Q|\!) \rangle$ is a prudent system $(\mathfrak{i}, \mathtt{n}) \notin \widehat{t'}$ then*

$$\Gamma \vdash P \mid R \sqsubseteq_{\mathbf{M}} \Gamma \vdash \langle P \mid (\!|Q|\!) \rangle \mid R$$

*for all $R \in \mathcal{P}$.*

*Proof.* We prove that for all observer $O$

$$\Gamma \vdash P \mid R \text{ must } O \implies \langle P \mid (\!|Q|\!) \rangle \mid R \text{ must } O \tag{10}$$

If $O \searrow \checkmark$ or $P \searrow \mathtt{err}$ the proposition holds immediately. Otherwise we proceed by induction on the transition steps. $P \mid R \parallel O$ cannot move because of `(os-tick)` since $O \not\searrow \checkmark$. $P \mid R \parallel O$ can make the following steps:

**(os-comm)** In this case $P \mid R = \mathbf{C}[\pi.P_1]$ and $\Gamma \vdash \mathbf{C}[\pi.P_1] \parallel O \rightsquigarrow \Gamma \vdash \mathbf{C}[P_1] \parallel O$ then, also by (os-comm), we can set $\langle P \mid (\!|Q|\!)\rangle \mid R = \mathbf{C}'[\pi.P_1]$ and $\Gamma \vdash \mathbf{C}'[\pi.P_1] \parallel O \rightsquigarrow \Gamma \vdash \mathbf{C}'[P_1] \parallel O$. The proposition holds by induction.

**(os-fail-in)** The case is similar to the case for (os-comm).

**(os-fail-out)** In this case the observed system either moves to a state $\Gamma \vdash P' \mid R \parallel O'$ or to the state $\Gamma \vdash P \mid R' \parallel O'$. In the first case process $P'$ is erroneous and in the second case or $R'$ is erroneous. In the first case Equation (11) is true since $\Gamma \vdash P' \mid R \,\text{MUST}\, O'$ does not hold, in the second case because $\Gamma \vdash P \mid R' \,\text{MUST}\, O'$ does not hold.

**(os-sum)/(os-cong)** The properties hold similarly to the previous three cases.

**(os-s)** If $\Gamma \vdash P \mid R \rightsquigarrow S'$ by:

- (ntx1) then $S' = \Gamma \vdash \mathbf{C}[\mathtt{err}]$. Since the process in the system is erroneous then $S' \,\text{MUST}\, O$ does not hold and the proposition holds immediately for this case.
- (ntx2) then $P \mid R = \mathbf{C}[s \,\&\!\!-\, A.P']$ with $\mathbf{C}[\square]$ s-a and $\Gamma \vdash P \mid R \rightsquigarrow \Gamma\mathbf{C}[P'] \mid P_S$ where $\Gamma(s,a) = P_S$ for $a \in A$. Attribute $a$ can be either s or ns (not n by hypothesis[7] On the other hand, $\langle P \mid (\!|Q|\!)\rangle \mid R$ can make different moves according to whether the service call is in $P$ or in $R$. If the service call is in $R$ then $\langle P \mid (\!|Q|\!)\rangle \mid R = \mathbf{C}'[s \,\&\!\!-\, A.P']$ with $\mathbf{C}'[\square]$ is s-a. In this case $\Gamma \vdash P \rightsquigarrow \Gamma\mathbf{C}'[P'] \mid P_S$ by (ntx2) and the proposition holds by induction since the hypothesis are preserved for the continuations. If the service call is in $P$ then $\langle P \mid (\!|Q|\!)\rangle \mid R = \mathbf{C}'[\langle s \,\&\!\!-\, A.P' \mid (\!|Q|\!)\rangle]$. In this case $\mathbf{C}'[\langle s \,\&\!\!-\, A.P' \mid (\!|Q|\!)\rangle]$ can move either by (tx1) or by (tx3) according to whether $a = \mathtt{s}$ or $a = \mathtt{n}s$. If $a = \mathtt{s}$ then $\Gamma \vdash \mathbf{C}'[\langle s \,\&\!\!-\, A.P' \mid (\!|Q|\!)\rangle] \rightsquigarrow \Gamma \vdash \mathbf{C}'[\langle P' \mid P_S \mid (\!|Q|\!)\rangle]$ by (tx1). In this case the proposition holds by induction ($\Gamma \vdash (P' \mid P_S) \mid R \,\text{MUST}\, \Gamma \vdash \langle P' \mid P_S \mid (\!|Q|\!)\rangle \mid R$) since the hypothesis are preserved. If $a = \mathtt{ns}$ then $\Gamma \vdash \mathbf{C}'[\langle s \,\&\!\!-\, A.P' \mid (\!|Q|\!)\rangle] \rightsquigarrow \Gamma \vdash \mathbf{C}'[\langle s \,\&\!\!-\, A.P' \mid (\!|Q|\!)\rangle \mid P_S]$. In this case the proposition holds by induction ($\Gamma \vdash P' \mid (R \mid P_S) \,\text{MUST}\, \Gamma \vdash \langle P' \mid (\!|Q|\!)\rangle \mid (R \mid P_S)$ since the hypothesis are preserved.
- (ntx3) then $P \mid R = \mathbf{C}[s \,\&\!\!-\, A.P']$ with $\mathbf{C}[\square]$ s-a and $\Gamma \vdash P \mid R \rightsquigarrow \Gamma\mathbf{C}[P'] \mid \langle P_S \rangle$ where $\Gamma(s,a) = P_S$ for $a \in A$. Attribute $a$ is r$n$ (it cannot be r by hypothesis). Since $a = \mathtt{rn}$ then $\langle P \mid (\!|Q|\!)\rangle \mid R = \mathbf{C}'[\langle s \,\&\!\!-\, A.P' \mid (\!|Q|\!)\rangle]$ and $\Gamma \vdash \langle P \mid (\!|Q|\!)\rangle \mid R \rightsquigarrow \Gamma \vdash \mathbf{C}'[\langle P' \mid (\!|Q|\!)\rangle] \mid \langle P_S \rangle$. In this case the proposition holds by induction ($\Gamma \vdash P' \mid (R \mid \langle P_S \rangle) \,\text{MUST}\, \Gamma \vdash \langle P' \mid (\!|Q|\!)\rangle \mid (R \mid \langle P_S \rangle)$ since the hypothesis are preserved.
- (tx1)/(tx2)/(tx3)/(tx3)/(tx4) then $P = \mathbf{C}[P_\mathrm{i}]$ for some $P_\mathrm{i}$. We can also apply the same rule to $\langle P \mid (\!|Q|\!)\rangle$ with $\mathbf{C}'[\square] = \langle \mathbf{C}[\square] \mid (\!|Q|\!)\rangle$. The theorem holds by induction.
- (s-p) then $P$ moves for one of the rule for processes in Definition 4. $\langle P \mid (\!|Q|\!)\rangle$ can do the same step with context $\langle \square \mid (\!|Q|\!)\rangle$ and the theorem holds by induction.

□

[7] We could remove this hypothesis by requiring a compensation that, instead of actually compensating, continues the activity that had been suspended by the failure of the running process.

## 8 Concluding Remarks and Related Work

We embed a few primitives for managing the dynamic reconfiguration of transactional scopes in ATc to generalise the transactional mechanisms of EJB to SOC so to have consistent and predictable failure propagation. We give a type system that guarantees absence of failures due to misuse of transactional attributes. The adopted mechanisms find their basic motivations in SOC where one open issue is the lack of agreement on the semantics of dynamic reconfigurations of transactional scopes. Such problem is amplified when services support and rely on different kinds of transactional behaviour. An original contribution of this paper is the definition of mechanisms to *determine* and *control* the dynamic reconfiguration of distributed transactions. Service invocations cause systems reconfiguration; specifically, invocations may dynamically introduce new transactional scopes or rearrange the old ones.

We are currently extending ATc with a theory of testing [15] where observers can cause communication failures. The aim is to test the correctness of the system behaviour, including failure handling and compensations. On this basis it is possible to define a notion of equivalence for ATc systems. The intuition is that two systems are equivalent if they satisfy the same set of tests; some preliminary results are summarised below (the interested reader is referred to [5] for a detailed presentation).

The theory of testing of ATc shows that under some conditions some transactional attributes are equivalent. Namely, it is possible to replace a transactional attribute with an equivalent one without altering the behaviour of the system. Notice that this also allows one to specify a larger set of transactional attributes for service invocations. For example, $\langle s \,\mathfrak{E}\!\vdash A.P \mid (\!|Q|\!)\rangle$ maintain the same behaviour if $A$ is any of the subsets of $\{\mathtt{r}, \mathtt{m}, \mathtt{s}\}$ since the invocation of $s$ happens inside a transaction.

Since both dynamic reconfiguration and LRT are a key aspects in SOC, it is crucial to provide a formal account of their inter-relationships and to understand and control the mechanisms of failure.

Languages for service orchestration (e.g., WS-BPEL [17]) provide support for distributed transactions and have been modelled extending some process calculi like those in [3, 10, 12, 13] with primitives that allow a party to define the scopes, failure handlers, and compensation mechanisms (see [20] for an overview and a comparison of such approaches). StAC [8] and CJoin [6] are process calculi which model arbitrarily nested transactions and focus on the separation of process management with error/compensation. The latter offers a mechanism to merge different scopes but it is not offering the flexibility of the transactional attributes of ATc.

At the best of our knowledge, none of the proposed framework has been given a type system as the one proposed here (a formal comparison of different approaches for compensations in flow composition languages can be found in [7]). The existing literature addresses only part of the dynamic aspects involved in error management. For example, [11] proposes a model for dynamic installations of compensation processes, however, dynamic reconfigurations of transactional scopes have not been considered.

One of the limitations of our approach is the lack of link mobility à la $\pi$-calculus; the extension of our approach to a name passing calculus is left as future work. We argue that the type discipline proposed here can be simply adapted to a name passing version of ATc. In fact, our type system is orthogonal to the communication mechanisms. On

the contrary, the testing theory of ATc will be greatly affected by the introduction of name passing features.

Other interesting extensions would be to allow the communication of attributes and a primitive enabling a service $s$ to make a parametrised invocation to a service $s'$ using the same attribute supporte by $s$ (recall that attributes are when services are published in containers). Such features give a great expressiveness but require more sophisticated type disciplines.

An orthogonal topic is the modelling of protocols for deciding the outcome of distributed transactions (e.g., the work in [1]). Some standards like Business Transaction Protocol (BTP) [16] and Web Service Transaction (WS-Tx [18]) have been proposed for LRTs. Such protocols involve a more general scenario than the classic *atomic commit*: the global consensus is no longer necessary and is substituted by weaker constraints. In [2, 4] BTP cohesion along with the properties ensured by the "weakened" constraints have been studied via a formalisation in the asynchronous $\pi$-calculus (see [9] for an overview on the *cohesion*-base approach of BTP).

The present paper provides a high level semantics of failure propagation, compensation and scope reconfiguration, while abstracting from protocols necessary to implement them. Consider, for example, the process $\langle s \,\&\!\!\leftarrow\, \{\mathtt{r}\}.P \mid (\!|Q|\!)\rangle$ invoking a service $s$ whose body is $x[\![P']\!].Q'$. Since service $s$ supports the attribute $\mathtt{r}$, its body is executed inside the same scope (if any) of the caller, according to Definition 6.

$$\Gamma \vdash \langle s \,\&\!\!\leftarrow\, \{\mathtt{r}\}.P \mid (\!|Q|\!)\rangle \rightsquigarrow^* \Gamma \vdash \langle P \mid P' \mid (\!|Q \mid Q'|\!)\rangle$$

The same above includes compensations of different possibly cross-domain and distributed processes. Noteworthy, the mechanism that trigger $Q$ and $Q'$ are not trivial The higher level perspective we adopted has the advantage of providing a concise but rigorous understanding of dynamic scope reconfigurations. We leave the investigation of the underneath coordination protocols, which would provide a skeleton for the implementation of the higher level mechanisms, as a future work. (We remark that this issue is common to any theory of distributed transactions.)

## References


1. M. Berger and K. Honda. The two-phase commitment protocol in an extended pi-calculus. *Electr. Notes Theor. Comput. Sci.*, 39(1), 2000.
2. L. Bocchi. Compositional nested long running transactions. In *FASE*, volume 2984 of *LNCS*, pages 194–208. Springer, 2004.
3. L. Bocchi, C. Laneve, and G. Zavattaro. A calculus for long-running transactions. In E. Najm, U. Nestmann, and P. Stevens, editors, *FMOODS*, volume 2884 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 2003.
4. L. Bocchi and R. Lucchi. Atomic commit and negotiation in service oriented computing. In *COORDINATION*, volume 4038 of *LNCS*, pages 16–27. Springer, 2006.
5. L. Bocchi and E. Tuosto. A Java Inspired Semantics for Transactions in SOC (extended report), 2009. Available at `http://www.cs.le.ac.uk/people/lb148/javatransactions.html`.
6. R. Bruni, H. Melgratti, and U. Montanari. Nested Commits for Mobile Calculi: Extending Join. In J.-J. Lévy, E. Mayr, and J. Mitchell, editors, *IFIP TCS*, pages 563–576, 2004.

7. R. Bruni, H. C. Melgratti, and U. Montanari. Theoretical foundations for compensations in flow composition languages. In *POPL*, pages 209–220. ACM, 2005.
8. M. Butler and C. Ferreira. An operational semantics for stac, a language for modelling long-running business transactions. In *In Coordination 2004, volume 2949 of LNCS*, pages 87–104. Springer-Verlag, 2004.
9. S. Dalal, S. Temel, M. Little, M. Potts, and J. Webber. Coordinating business transactions on the web. *IEEE Internet Computing*, 7(1):30–39, 2003.
10. C. Guidi, I. Lanese, F. Montesi, and G. Zavattaro. On the interplay between fault handling and request-response service invocations. In *ACSD*, pages 190–198. IEEE, 2008.
11. C. Guidi, I. Lanese, F. Montesi, and G. Zavattaro. Dynamic error handling in service oriented applications. *Fundam. Inf.*, 95(1):73–102, 2009.
12. C. Laneve and G. Zavattaro. Foundations of web transactions. In *FoSSaCS*, volume 3441 of *LNCS*, pages 282–298. Springer, 2005.
13. M. Mazzara and I. Lanese. Towards a unifying theory for web services composition. In *WS-FM*, volume 4184 of *LNCS*, pages 257–272. Springer, 2006.
14. S. Microsystems. Enterprise javabeans (ejb) technology, 2009. `http://java.sun.com/products/ejb/`.
15. R. D. Nicola and M. C. B. Hennessy. Testing equivalences for processes. *Theoretical Comput. Sci.*, 34(1–2):83–133, Nov. 1984.
16. OASIS. Business Transaction Protocol (BTP), 2002.
17. OASIS. Web Services Business Process Execution Language (WS-BPEL) Version 2.0. Technical report, 2007.
18. OASIS. Web Services Transaction (WS-TX), 2009.
19. D. Panda, R. Rahman, and D. Lane. *EJB3 in action*. Manning, 2007.
20. C. Vaz, C. Ferreira, and A. Ravara. Dynamic recovering of long running transactions. In *TGC*, volume 5474 of *LNCS*, pages 201–215. Springer, 2008.

# A Proofs of the Results in § 4

**Proposition 1.** *The operator $\_ \oplus \_$ is idempotent, associative and commutative.*

*Proof.* Let $|\_|$ be the function mapping types to natural numbers defined as

$$|0| = 0 \qquad |(I, t_1, t_2)| = 1 + max\{|t_1|, |t_2|\}$$

We prove that $t \oplus t = t$ by induction on $|t|$. The base case is trivial since $|t| = 0$ iff $t = 0$. Assume $\oplus$ idempotent on any type $t'$ such that $|t'| < |t|$, then $t = (I, t_1, t_2)$ for some $I$, $t_1$ and $t_2$. By definition, $t \oplus t = (I \cup I, t_1 \oplus t_1, t_2 \oplus t_2)$ and, by inductive hypothesis and idempotency of $\cup$, we conclude $t \oplus t = t$.

The proofs for associativity and commutativity are similar. □

**Proposition 2.** *All the operators $\_^{\Subset}$, $\_^{\downarrow}$, and $\_^{?}$ distribute over $\_ \oplus \_$. Moreover, $(t_1 \oplus t_2)^{\Subset} = {t_1}^{\Subset} \cup {t_2}^{\Subset}$.*

*Proof.* Trivially, from the Definition 7.

For convinience we recall here Definition 8.

**Definition 8.** *The typing rules for non-erroneous processes (cf. Definition 5) are*

$$\texttt{(idle)}\frac{}{0 \triangleright 0} \qquad \texttt{(res)}\frac{P \triangleright t}{\nu x\, P \triangleright t} \qquad \frac{P \triangleright t \qquad P' \triangleright t'}{P \mid P' \triangleright t \oplus t'}\,\texttt{(par)} \qquad \frac{P \triangleright t}{!P \triangleright t}\,\texttt{(repl)}$$

$$\texttt{(inv)}\frac{P \triangleright t_p \qquad I = \{\mathfrak{o}\} \times A}{s \Subset A.P \triangleright (I \cup {t_p}^{\Subset}, {t_p}^{\downarrow}, {t_p}^{?})} \qquad \frac{P \triangleright t_p \qquad Q \triangleright t_q}{\pi[\![Q]\!].P \triangleright ({t_p}^{\Subset}, {t_p}^{\downarrow}, t_q \oplus {t_p}^{?})}\,\texttt{(comp)}$$

$$\texttt{(scope}_1\texttt{)}\frac{P \triangleright (I, t_c, t_u) \qquad Q \triangleright t_q}{\langle P \mid (\!|Q|\!)\rangle \triangleright ((I \cup {t_c}^{\Subset})[\![\mathfrak{o} \mapsto \mathfrak{i}]\!], t_u \oplus {t_c}^{\downarrow} \oplus {t_c}^{?} \oplus t_q, 0)} \qquad \frac{P \triangleright 0}{\langle P \mid (\!|Q|\!)\rangle \triangleright 0}\,\texttt{(scope}_2\texttt{)}$$

*where, for $I \subset \{\mathfrak{i}, \mathfrak{o}\} \times \mathcal{A}$, $I[\![\mathfrak{o} \mapsto \mathfrak{i}]\!] \stackrel{def}{=} \{(\mathfrak{i}, a) : (\mathfrak{o}, a) \in I\} \cup (I \cap \{\mathfrak{i}\} \times \mathcal{A})$.*

**Proposition 3.** *For each non-erroneous $P \in \mathcal{P}$ there is a unique type $t$ such that $P \triangleright t$.*

*Proof.* By induction on the structure of $P$ and inspection of the typing rules in Definition 8. □

**Proposition 4.** *For any non-erroneous $P, Q \in \mathcal{P}$, if $P \equiv Q$ and $P \triangleright t$ then $Q \triangleright t$.*

*Proof.* The proof easily follows by induction on the derivation of $P \equiv Q$ (considering Definition 2 as the specification of a proof system).

Rule `(par)`, the definition of $\oplus$, and Proposition 1 prove the thesis if $P \equiv Q$ is obtained from the monoidal laws of the parallel composition.

The axioms on restriction are accommodated by rule `(res)` (e.g., if $P \triangleright t$ then $\nu x\, P \triangleright t$, hence $\nu y\, \nu x\, P \triangleright t$ and similarly for the other side of $\nu y\, \nu x\, P \equiv \nu x\, \nu y\, P$).

Rules `(repl/par)` proved that if $P \triangleright t$ then $!P \mid P \triangleright t \oplus t$ and, by idempotency of $\oplus$ we have the thesis.

Let $\langle P \mid (\!|R|\!)\rangle \equiv \langle Q \mid (\!|R|\!)\rangle$ be derived by $P \equiv Q$ and $P \triangleright t$. By inductive hypothesis, $Q \triangleright t$, hence the thesis follows by rule $\texttt{(scope}_1\texttt{)}$.

The other cases are similar. □

Some auxiliary lemmas are necessary before proving the correctness of our typing discipline (Theorem 1 and Corollary 1).

**Lemma 1.** *Let $P \in \mathcal{P}$ be well-typed and $\mathbf{C}[\boxminus]$ be a context. If $\mathbf{C}[P]$ is well-typed then any transactional scope which is a subterm of $\mathbf{C}[0]$ is well-typed or it is nested in a well-typed transactional scope in $\mathbf{C}[\boxminus]$.*

*Proof.* The thesis is by induction on the structure of the context $\mathbf{C}[\boxminus]$.

The base cases $\mathbf{C}[\boxminus] = \boxminus$ and $\mathbf{C}[\boxminus] = 0$ are trivial.

Assume $\mathbf{C}[\boxminus] = Q \mid \mathbf{C}'[\boxminus]$, then the transactional scopes in $\mathbf{C}[\boxminus]$ are subterms either of $Q$ or of $\mathbf{C}'[\boxminus]$. Since $\mathbf{C}[P]$ is well-typed both $Q$ and $\mathbf{C}'[P]$ are well-typed (otherwise by rule (par) we get a contradiction). Hence the inductive hypothesis guarantees that any transactional scope of $Q$ or $\mathbf{C}'[0]$ is well-typed or nested in a well-typed scope. (The proof for $\mathbf{C}[\boxminus] = \mathbf{C}'[\boxminus] \mid Q$ is analogous).

If $\mathbf{C}[\boxminus] = \langle \boxminus \mid Q \mid (\!|R|\!) \rangle$, the well-typedness of $\mathbf{C}[P]$ implies that $\langle Q \mid (\!|R|\!) \rangle$ is well-typed (otherwise rule ($\mathtt{scope}_1$) yields a contradiction); by inductive hypothesis, the statement holds for the scopes is $Q$ or $R$. □

Noticing that $\mathbf{C}[0]$ is well-typed if $\mathbf{C}[P]$ is well-typed for a well-typed $P \in \mathcal{P}$, Lemma 1 can be restates as

**Corollary 2.** *If $\mathbf{C}[\boxminus]$ is a context such that $\mathbf{C}[0]$ is well-typed, then any transactional scope subterm of $\mathbf{C}[0]$ is well-typed or it is nested in a well-typed transactional scope in $\mathbf{C}[\boxminus]$.*

**Lemma 2.** *If $\langle P \mid (\!|Q|\!) \rangle$ is well-typed then, for any context $\mathbf{C}[\boxminus]$ such that $\mathbf{C}[0]$ is well-typed, $\mathbf{C}[\langle P \mid (\!|Q|\!) \rangle]$ is well-typed.*

*Proof.* Trivially, by structural induction on the structure of $\mathbf{C}[\boxminus]$. □

Remarkably, Lemma 2 does not hold for generic well-typed processes (e.g., consider the process of Example 9).

**Lemma 3.** *If $P, Q \in \mathcal{P}$ are well-typed then $P \mid Q$ is well-typed.*

*Proof.* Let $P \triangleright t_p$ and $Q \triangleright t_q$, by rule (par), $P \mid Q \triangleright t_p \oplus t_q$. The thesis follows by observing that $\widehat{t_p \oplus t_q} = \widehat{t_p} \cup \widehat{t_q}$. □

By inspection of the rule ($\mathtt{scope}_1$), a transactional scope may contain the pair $(\mathfrak{o}, \mathfrak{m})$ only in its second component (the left child of the type). This observation is used in Lemma 4 below.

**Lemma 4.** *If $\langle P_1 \mid (\!|Q_1|\!) \rangle$ and $\langle P_2 \mid (\!|Q_2|\!) \rangle$ are well-typed then $\langle P_1 \mid P_2 \mid (\!|Q_1 \mid Q_2|\!) \rangle$ is well-typed.*

For convinience we repeat Definition 6.

**Definition 6.** *The* reduction relation of ATc systems *is the smallest relation* $\rightsquigarrow$ *closed under the following rule and axiom:*

$$\texttt{(ntx1)} \quad \Gamma \vdash \mathbf{C}[s \,\&\!\!\leftarrow A.P] \rightsquigarrow \Gamma \vdash \texttt{err} \qquad \mathtt{m} \in A$$

$$\texttt{(ntx2)} \quad \Gamma \vdash \mathbf{C}[s \,\&\!\!\leftarrow A.P] \rightsquigarrow \Gamma \vdash \mathbf{C}[P] \mid R \qquad R \in \Gamma(s, \{\mathtt{s}, \mathtt{n}, \mathtt{ns}\} \cap A)$$

$$\texttt{(ntx3)} \quad \Gamma \vdash \mathbf{C}[s \,\&\!\!\leftarrow A.P] \rightsquigarrow \Gamma \vdash \mathbf{C}[P] \mid \langle R \rangle \qquad R \in \Gamma(s, \{\mathtt{r}, \mathtt{rn}\} \cap A)$$

$$\texttt{(tx1)} \quad \frac{P = \mathbf{C}[\langle s \,\&\!\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q|\!) \rangle] \qquad \mathit{bc}(P) \cap \mathit{fc}(R) = \emptyset}{\Gamma \vdash P \rightsquigarrow \Gamma \vdash \mathbf{C}[\langle P_1 \mid P_2 \mid R \mid (\!|Q|\!) \rangle]} \qquad R \in \Gamma(s, \{\mathtt{m}, \mathtt{s}, \mathtt{r}\} \cap A)$$

$$\texttt{(tx2)} \quad \Gamma \vdash \mathbf{C}[\langle s \,\&\!\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q|\!) \rangle] \rightsquigarrow \Gamma \vdash \mathbf{C}[Q] \qquad \mathtt{n} \in A$$

$$\texttt{(tx3)} \quad \Gamma \vdash \mathbf{C}[\langle s \,\&\!\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q|\!) \rangle] \rightsquigarrow \Gamma \vdash \mathbf{C}[\langle P_1 \mid P_2 \mid (\!|Q|\!) \rangle] \mid R \qquad \mathtt{ns} \in A \wedge R \in \Gamma(s, \mathtt{ns})$$

$$\texttt{(tx4)} \quad \Gamma \vdash \mathbf{C}[\langle s \,\&\!\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q|\!) \rangle] \rightsquigarrow \Gamma \vdash \mathbf{C}[\langle P_1 \mid P_2 \mid (\!|Q|\!) \rangle] \mid \langle R \rangle \qquad \mathtt{rn} \in A \wedge R \in \Gamma(s, \mathtt{rn})$$

$$\texttt{(s-p)} \quad \frac{P \to P'}{\Gamma \vdash P \rightsquigarrow \Gamma \vdash P'}$$

*where* $\mathbf{C}[\square] \neq 0$ *and* $\mathbf{C}[\square]$ *is s-a in* (ntx1 ÷ 3).

**Theorem 1.** *Let* $P \in \mathcal{P}$ *be well-typed. For every well-typed environment* $\Gamma$*, if* $\Gamma \vdash P \rightsquigarrow \Gamma \vdash Q$ *then* $Q$ *is well-typed.*

*Proof.* Let $P \triangleright t$ we proceed by case analysis on transactionality of invocations.

For non-transactional invocations, let $\mathbf{C}[\square]$ be a s-a context such that $P = \mathbf{C}[s \,\&\!\!\leftarrow A.P']$.

**(ntx1)** The transition in the statement cannot be derived using this axiom otherwise we would have $(\mathfrak{o}, \mathtt{m}) \in \widehat{t}$ which contradicts well-typedness of $P$ (as $\mathbf{C}[\square]$ is s-a).

**(ntx2)** In this case, $Q = \mathbf{C}[P'] \mid R$ where $R$ is the body of a service $s$ in $\Gamma$ supporting an attribute $a \in \{\mathtt{s}, \mathtt{n}, \mathtt{ns}\}$. Assume $R \triangleright t_r$, by (4) in Definition 11 and well-typedness of $\Gamma$, we have that $(\mathfrak{o}, \mathtt{m}) \notin \widehat{t_r}$. Hence, by the typing rule (par) (and possibly (repl)) $\mathbf{C}[P' \mid R]$ is well-typed since $\mathbf{C}[\square]$ may only consists of parallel and replicated contexts.

**(ntx3)** This case is analogous to the previous one but for the fact that (3) in Definition 11 is used to prove well-typedness of the body of the service.

For transactional invocations, let $\mathbf{C}[\square]$ be a context such that $P = \mathbf{C}[\langle s \,\&\!\!\leftarrow A.P_1 \mid P_2 \mid (\!|Q|\!) \rangle]$.

**(tx1)** Assume that

- $R$ is the body of a service $s$ in $\Gamma$ supporting an attribute $a \in \{\mathtt{m}, \mathtt{s}, \mathtt{r}\}$ and $R \triangleright t_r$;
- $Q \triangleright t_q$, $P_1 \triangleright t_1$, $P_2 \triangleright t_2$, and $\mathbf{C}[\langle P_1 \mid P_2 \mid R \mid (\!|Q|\!) \rangle] \triangleright t'$.

The proof is by induction on the structure of $\mathbf{C}[\square]$.

If $\mathbf{C}[\square] = \square$ then the flat type of $\langle P_1 \mid P_2 \mid (\!|Q|\!) \rangle$ is included in $\widehat{t}$ and therefore $\langle P_1 \mid P_2 \mid (\!|Q|\!) \rangle$ is well-typed. By hypothesis, $\langle R \rangle$ is well-typed then $\langle P_1 \mid R \mid P_2 \mid (\!|Q|\!) \rangle$ is well-typed by Lemma 4.

Let $\mathbf{C}[\square] = \langle \square \mid P_0 \mid (\!|Q_0|\!) \rangle$; by Corollary 2 $\mathbf{C}[0]$ well-typed. Also, observe that the flat type of $\langle P_1 \mid R \mid P_2 \mid (\!|Q|\!) \rangle$ is a subset of $\widehat{t^{\downarrow}}$ plus $t_r{}^{\downarrow}$ and both such sets do not contain $(\mathfrak{o}, \mathtt{m})$.

**(tx2)** Let $Q \triangleright t_q$; we proceed by induction on the structure of $\mathbf{C}[\square]$.
If $\mathbf{C}[\square] = \square$, it suffices to observe that $\widehat{t_q} \subseteq \widehat{t}$, hence $(\mathfrak{o}, \mathtt{m}) \notin \widehat{t_q}$.
If $\mathbf{C}[\square] = R \mid \mathbf{C}'[\square]$, then both $R$ and $\mathbf{C}'[\langle s \,\&\!\!\leftarrow\, A.P_1 \mid P_2 \mid (\!|Q|\!)\rangle]$ must be well-typed (otherwise rule `(par)` would yield a contradiction). By inductive hypothesis, $\mathbf{C}'[Q]$ is well-typed and therefore $\mathbf{C}[Q] = R \mid \mathbf{C}'[Q]$ is well-typed (Lemma 3). The proof is similar for $\mathbf{C}[\square] = R \mid \mathbf{C}'[\square] \mid R$.

**(tx3)** Since $P$ is of the form $\mathbf{C}[\langle s \,\&\!\!\leftarrow\, A.P_1 \mid P_2 \mid (\!|Q|\!)\rangle]$, its flat type includes the flat type of $\mathbf{C}[\langle P_1 \mid P_2 \mid (\!|Q|\!)\rangle]$ (as, by rule `(inv)`, the former is obtained by adding to the latter the modality/attribute pairs of the invocation to $s$). Hence, $\mathbf{C}[\langle P_1 \mid P_2 \mid (\!|Q|\!)\rangle]$ is well-typed, moreover $R$ is the body of a service supporting `ns` and is assumed well-typed in $\Gamma$. Therefore, $\mathbf{C}[\langle P_1 \mid P_2 \mid (\!|Q|\!)\rangle] \mid R$ is also well-typed because (by Lemma 3).

**(tx4)** Similar to `(tx3)`.

□

## B Auxiliary Definitions

Formally, we consider attribute substitutions as functions that maps processes to processes (and (observed) systems to (observed) systems). A (standard) attribute substitution is denoted as $[^b/_a]$ and it is meant to replace $a$ with $b$ in a process $P$ as follows:

$$
\begin{array}{ll}
\mathtt{0}[^b/_a] & = \mathtt{0} \\
(\nu x\ P)[^b/_a] & = \nu x\ (P[^b/_a]) \\
(P \mid Q)[^b/_a] & = P[^b/_a] \mid Q[^b/_a] \\
(!P)[^b/_a] & = !(P[^b/_a]) \\
(s \,\&\!\!\leftarrow\, A.P)[^b/_a] & = s \,\&\!\!\leftarrow\, A \setminus \{a\} \cup \{b\}.(P[^b/_a]) \\
(\langle P \mid (\!|Q|\!)\rangle)[^b/_a] & = \langle P[^b/_a] \mid (\!|Q[^b/_a]|\!)\rangle \\
(\pi[\![Q]\!].P)[^b/_a] & = \pi[\![Q[^b/_a]]\!].(P[^b/_a]) \\
\mathtt{err}[^b/_a] & = \mathtt{err}
\end{array}
$$

To avoid cumbersome parenthesis, substitutions are supposed to have precedence on all syntactic operators.

We consider two special classes of substitutions $[^-/_-]^{\mathfrak{o}}$ and $[^-/_-]^{\mathfrak{i}}$ that allow us to replace attributes respectively outside and inside transactional scopes.

**Definition 7.** *The substitutions $P[^b/_a]^{\mathfrak{o}}$ and $P[^b/_a]^{\mathfrak{i}}$ are defined by induction on the structure of the $P$:*

$$
\begin{array}{ll}
0[^b/_a]^{\mathfrak{o}} & = 0 \\
\nu x\ P[^b/_a]^{\mathfrak{o}} & = \nu x\ (P[^b/_a]^{\mathfrak{o}}) \\
(P \mid Q)[^b/_a]^{\mathfrak{o}} & = P[^b/_a]^{\mathfrak{o}} \mid Q[^b/_a]^{\mathfrak{o}} \\
(!P)[^b/_a]^{\mathfrak{o}} & = !P[^b/_a]^{\mathfrak{o}} \\
(s \,\&\!\!\leftarrow\, A.P)[^b/_a]^{\mathfrak{o}} & = s \,\&\!\!\leftarrow\, A \setminus \{a\} \cup \{b\}.P[^b/_a]^{\mathfrak{o}} \\
\langle P \mid (\!|Q|\!)\rangle[^b/_a]^{\mathfrak{o}} & = \langle P \mid (\!|Q[^b/_a]^{\mathfrak{o}}|\!)\rangle \\
(\pi[\![Q]\!].P)[^b/_a]^{\mathfrak{o}} & = \pi[\![Q[^b/_a]^{\mathfrak{o}}]\!].P[^b/_a]^{\mathfrak{o}}
\end{array}
\qquad
\begin{array}{ll}
0[^b/_a]^{\mathfrak{i}} & = 0 \\
(\nu x\ P)[^b/_a]^{\mathfrak{i}} & = \nu x\ P[^b/_a]^{\mathfrak{i}} \\
(P \mid Q)[^b/_a]^{\mathfrak{i}} & = P[^b/_a]^{\mathfrak{i}} \mid Q[^b/_a]^{\mathfrak{i}} \\
(!P)[^b/_a]^{\mathfrak{i}} & = !P[^b/_a]^{\mathfrak{i}} \\
(s \,\&\!\!\leftarrow\, A.P)[^b/_a]^{\mathfrak{i}} & = s \,\&\!\!\leftarrow\, A.(P[^b/_a]^{\mathfrak{i}}) \\
\langle P \mid (\!|Q|\!)\rangle[^b/_a]^{\mathfrak{i}} & = \langle P[^b/_a] \mid (\!|Q[^b/_a]^{\mathfrak{i}}|\!)\rangle \\
(\pi[\![Q]\!].P)[^b/_a]^{\mathfrak{i}} & = \pi[\![Q[^b/_a]^{\mathfrak{i}}]\!].P[^b/_a]^{\mathfrak{i}}
\end{array}
$$

Let $\sigma$ denote any attribute substitution (standard or not).

**Proposition 5.** *Attribute substitutions are idempotent.*

*Proof.* The idempotency of $[^{-}/_{-}]^{\mathrm{o}}$ and $[^{-}/_{-}]^{\mathrm{i}}$ descends from the idempotency of $[^{-}/_{-}]$. The idempotency of $[^{-}/_{-}]$ is trivially obtained by induction on the structure of processes. □

Attribute substitutions can be generalised to contexts.

**Definition 8.** *A* generalised context *is a term derivable from the following grammar:*

$$\mathbf{C}[\natural] ::= \natural\sigma \;\Big|\; \mathit{0} \;\Big|\; \langle \natural\sigma \mid P \mid (\!|Q|\!) \rangle \;\Big|\; P \mid \mathbf{C}[\natural] \;\Big|\; \mathbf{C}[\natural] \mid P$$

*A generalised context* $\mathbf{C}[\natural]$ *is* s-a *if there are no* $P, Q \in \mathcal{P}$ *and generalised context* $\mathbf{C}'[\natural]$ *such that* $\mathbf{C}[\natural] = \mathbf{C}'[\langle \natural \mid P \mid (\!|Q|\!) \rangle]$.

Notice that contexts as in Definition 3 are generalised contexts where all substitutions are identities (i.e., $[^{a}/_{a}]$ for an $a \in \mathcal{A}$). Moreover, generalised contexts are obtained by "percolating" attribute substitutions trough the structure of contexts as in Definition 3 according to the following definition.

**Definition 9.** *Given a contexts* $\mathbf{C}[\natural]$ *as in Definition 3 and an attribute substitution* $\sigma$*,* $\mathbf{C}[\natural]\sigma$ *is defined by induction on the structure of* $\mathbf{C}[\natural]$

$$\begin{array}{lll}
\mathbf{C}[\natural] = \natural & \Longrightarrow & \mathbf{C}[\natural]\sigma = \natural\sigma \\
\mathbf{C}[\natural] = \mathit{0} & \Longrightarrow & \mathbf{C}[\natural]\sigma = \mathit{0} \\
\mathbf{C}[\natural] = \langle \natural \mid P \mid (\!|Q|\!) \rangle & \Longrightarrow & \mathbf{C}[\natural] = \langle \natural\sigma \mid P\sigma \mid (\!|Q\sigma|\!) \rangle \\
\mathbf{C}[\natural] = P \mid \mathbf{C}'[\natural] & \Longrightarrow & \mathbf{C}[\natural]\sigma = P\sigma \mid \mathbf{C}'[\natural]\sigma \\
\mathbf{C}[\natural] = \mathbf{C}'[\natural] \mid P & \Longrightarrow & \mathbf{C}[\natural]\sigma = \mathbf{C}'[\natural]\sigma \mid P\sigma
\end{array}$$

We extend to generalised context the definitions of $[^{-}/_{-}]^{\mathrm{o}}$ and $[^{-}/_{-}]^{\mathrm{i}}$. Namely, we define attribute substitutions $\sigma^{\mathrm{o}}$ and $\sigma^{\mathrm{i}}$ as follows.

**Definition 10.** *Let* $\mathbf{C}[\natural]$ *be a contexts as per Definition 3 and* $\sigma$ *a substitution of the form* $P[^{b}/_{a}]^{\mathrm{o}}$ *or* $P[^{b}/_{a}]^{\mathrm{i}}$*, for some* $a, b \in \mathcal{A}$*. Then* $\mathbf{C}[\natural]\sigma$ *is defined as follows*

$$\mathbf{C}[\natural]\sigma = \begin{cases}
\natural\sigma, & \text{if} \quad \mathbf{C}[\natural] = \natural \\
\mathit{0}, & \text{if} \quad \mathbf{C}[\natural] = \mathit{0} \\
\langle \natural \mid P \mid (\!|Q\sigma|\!) \rangle, & \text{if} \quad \mathbf{C}[\natural] = \langle \natural \mid P \mid (\!|Q|\!) \rangle \wedge \sigma = [^{b}/_{a}]^{\mathrm{o}} \\
\langle \natural[^{b}/_{a}] \mid P[^{b}/_{a}] \mid (\!|Q\sigma|\!) \rangle, & \text{if} \quad \mathbf{C}[\natural] = \langle \natural \mid P \mid (\!|Q|\!) \rangle \wedge \sigma = [^{b}/_{a}]^{\mathrm{i}} \\
P\sigma \mid \mathbf{C}'[\natural]\sigma, & \text{if} \quad \mathbf{C}[\natural] = P \mid \mathbf{C}'[\natural] \\
\mathbf{C}'[\natural]\sigma \mid P\sigma, & \text{if} \quad \mathbf{C}[\natural] = \mathbf{C}'[\natural] \mid P
\end{cases}$$

It is convenient to adopt the usual notion of context composition $\mathbf{C}[\natural] \circ \mathbf{C}'[\natural] = \mathbf{C}[\mathbf{C}'[\natural]]$.

**Proposition 6.** *Let* $a, b \in \mathcal{A}$*. For any contexts* $\mathbf{C}[\natural]$ *and* $\mathbf{C}'[\natural]$

$$(\mathbf{C}[\natural] \circ \mathbf{C}'[\natural])[^{b}/_{a}] = (\mathbf{C}[\natural][^{b}/_{a}]) \circ (\mathbf{C}'[\natural][^{b}/_{a}])$$

$$(\mathbf{C}[\natural] \circ \mathbf{C}'[\natural])[^{b}/_{a}]^{\mathrm{i}} = (\mathbf{C}[\natural][^{b}/_{a}]^{\mathrm{i}}) \circ (\mathbf{C}'[\natural][^{b}/_{a}]^{\mathrm{i}})$$

$$\mathbf{C}[\natural] \textit{ s-a} \implies (\mathbf{C}[\natural] \circ \mathbf{C}'[\natural])[^{b}/_{a}]^{\mathrm{o}} = (\mathbf{C}[\natural][^{b}/_{a}]^{\mathrm{o}}) \circ (\mathbf{C}'[\natural][^{b}/_{a}]^{\mathrm{o}})$$

*Proof.* By induction on the structure of $\mathbf{C}[\square]$. □

The follwoing example shows that the last statement in Proposition 6 does not hold if $\mathbf{C}[\square]$ is not s-a.

*Example 13.* Let $\mathbf{C}[\square] = \langle \square \rangle$ and $\mathbf{C}'[\square] = \langle \square \mid (\!|Q|\!) \rangle$. By definition,

$$(\mathbf{C}[\square] \circ \mathbf{C}'[\square])[{}^{b}/_{a}]^{\mathrm{o}} = \langle\langle \square \mid (\!|s \ \mathfrak{E} \ \{a\}|\!) \rangle\rangle$$

while $(\mathbf{C}[\square][{}^{b}/_{a}]^{\mathrm{o}}) \circ (\mathbf{C}'[\square][{}^{b}/_{a}]^{\mathrm{o}}) = \langle\langle \square \mid (\!|s \ \mathfrak{E} \ \{b\}|\!) \rangle\rangle$.

Notice that Proposition 5 implies the following corollary.

**Corollary 3.** *For any context* $\mathbf{C}[\square]$ *and substitution* $\sigma$, $(\mathbf{C}[\square]\sigma)\sigma = \mathbf{C}[\square]\sigma$.

Given a substitution $[{}^{b}/_{a}]$ we define its inverse $[{}^{b}/_{a}]^{-g}$ as $[{}^{a}/_{b}]$ (and similarly for $[{}^{\text{-}}/_{\_}]^{\mathrm{o}}$ and $[{}^{\text{-}}/_{\_}]^{\mathrm{i}}$.

**Lemma 5.** *For every system* $S$ *and substitution* $[{}^{\text{-}}/_{\_}]$, $(S\sigma)\sigma^{-1} = S$ *(and consequently,* $(\mathbf{C}[\square]\sigma)\sigma^{-1} = \mathbf{C}[\square]$*).*

*Proof.* Structural induction on $S$. □