Math. Struct. in Comp. Science (2015), *vol.* 25, *pp.* 619–665. © Cambridge University Press 2014
doi:10.1017/S0960129512000904 First published online 10 November 2014

Attribute-based transactions in service oriented computing

LAURA BOCCHI[1] and EMILIO TUOSTO[2,†]

[1]*School of Computing, University of Kent, CT2 7NF, Canterbury, UK*
Email: L.Bocchi@kent.ac.uk
[2]*Department of Computer Science, University of Leicester, Leicester, UK*
Email: emilio@le.ac.uk

Received 10 January 2011; revised 16 December 2011

We present a theory for the design and verification of distributed transactions in dynamically reconfigurable systems. Despite several formal approaches have been proposed to study distributed transactional behaviours, the inter-relations between failure propagation and dynamic system reconfiguration still need investigation. We propose a formal model for transactions in service oriented architectures (SOAs) inspired by the attribute mechanisms of the Java Transaction API. Technically, we model services in ATc (after '*A*ttribute-based *T*ransactional *c*alculus'), a CCS-like process calculus where service declarations are decorated with a *transactional attribute*. Such attribute disciplines, upon service invocation, how the invoked service is executed with respect to the transactional scopes of the invoker. A type system ensures that well-typed ATc systems do not exhibit run-time errors due to misuse of the transactional mechanisms. Finally, we define a testing framework for distributed transactions in SOAs based on ATc and prove that under reasonable conditions some attributes are observationally indistinguishable.

1. Introduction

The *service oriented computing* (SOC) paradigm envisages distributed systems as loosely-coupled computational elements that dynamically discover and bind to each other. SOC has imposed to re-think (among other concepts) the notion of transaction. The long-lasting and cross-domain nature of SOC makes it unfeasible to adopt classic ACID transactions, which are implemented by locking the involved resources. The formal investigation of SOC transactions – often referred to as *long-running transactions* – has been a topic of focus in recent years (see Section 8 for a non-exhaustive overview). Central to this investigation is the notion of *compensation*, a weaker version of the classic rollback mechanism of database systems. Typically, each activity in a long-running transaction is associated with a compensation that is *installed* upon execution of that activity. Run-time failures of an activity inside a transaction are 'backwardly' propagated and trigger the compensations of the activities executed before.

Compensations have been studied in relation to the mechanisms of failure propagation, for instance to determine the order of execution of different compensations in a complex

† This work has been partially sponsored by the project Leverhulme Trust award 'Tracing Networks'.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

workflow as in (Bocchi *et al.* 2003; Bruni *et al.* 2005; Lanese 2010). The intuition is illustrated by the following example:

$$\langle \mathsf{send_email} \natural C.P \rangle \mid S \tag{1}$$

where a long-running transaction (represented by the angled brackets) executes the activity send_email, contextually installs a compensation C, and then behaves like P. The transaction runs concurrently and interacts with a – possibly remote – service S (e.g., a mail server). The interaction send_email between the transaction and S installs C in the transactional scope and leads (1) to the state (2)

$$\langle P \rangle_C \mid S' \tag{2}$$

where S' represents the state of S after receiving the email. In (2), a failure occurring in P would trigger the execution of the compensation C. Note that the activities of C are application-dependent, and that in our example C may not be able to roll-back send_email (because the mail could have already reached the mail server) but C could send a further message which asks to ignore the previous message, or pay a fee, etc.

We consider two different kinds of failures. The first arises from the misuse of transactional attributes while the second arises from communication failures. The latter class of failures is modelled by letting systems to interact with 'observers' that can force communication failures. Intuitively, observers model a faulty communication infrastructure. For instance, if P in (2) has the form $\mathsf{pay} \natural C'.Q$, the observer $\lightning \mathsf{pay}.O$ would cause the failure of action pay and lead the system (2) into the state

$$C \mid S' \tag{3}$$

where the compensation C is executed.

A service-oriented system possibly changes its configuration at each service invocation that adds new instances of invoked services to 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. For example, consider the system $\langle \mathsf{invoke}\ s.P \rangle_C$ consisting of a process that invokes a service s and then behaves like P within a transactional scope with compensation C. For the sake of this example, assume that the invocation triggers a (possibly remote) instance of the service s, say Q. The configuration of the system after the invocation of s can be shaped according to several possibilities: should $\langle \mathsf{invoke}\ s.P \rangle_C$ evolve so to include Q in the existing scope (i.e., $\langle P \mid Q \rangle_C$)? Should instead Q be running in a different scope (i.e., $\langle P \rangle_C \mid \langle Q \rangle$)? Should otherwise Q be executed outside any transactional scope (i.e., $\langle P \rangle_C \mid Q$)? Or raise an exception triggering the compensation C? Notice that each alternative is reasonable and has an impact on the semantics of failure propagation. For instance, if the invocation results in the transactional scope $\langle P \mid Q \rangle_C$ then a failure in Q would trigger the compensation C while, in case the invocation becomes $\langle P \rangle_C \mid Q$ a failure of Q would not trigger C.

In this paper, we study failure propagation in dynamically reconfiguring scenarios, specifically (SOAs). In order to discipline the reconfiguration of transactional scopes, we use mechanisms inspired by the container managed transactions (CMT) adopted by

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Enterprise Java Beans (EJB). In CMT, objects are published through *containers* (EJB 2009; Panda *et al.* 2007) specifying, for each method, an attribute that determines:

— the (transactional) requirements that the invoking party must satisfy (e.g., '*calling* `fooBar` *from outside a transactional scope throws an exception*'),
— how the transactional scopes dynamically reconfigure (e.g., '`fooBar` *is always executed in a newly created transactional scope*').

Attributes in CMT express only the requirements of the callee (i.e., invokers cannot specify requirements on the invoked method). This limitation is not desirable in SOC where service invocations are resolved at run-time by selecting one of the available implementations matching a given service description; typically, both the service requester and the service provider are allowed to express their requirements, which are matched when defining the service level agreement (SLA) for an invocation. Hence, it is natural in SOC to allow callers to express some requirements on the transactional behaviour of the invoked service (e.g., the invocation of s in $\langle \mathsf{invoke}\ s.P \rangle_C$ may require Q to be executed in the same transactional scope of P).

We argue that transactional attributes could let callers and callees to 'negotiate' the transactional behaviour resulting from their interaction. In this respect, transactional attributes can be considered as part of the SLA 'contract' between requester and provider. The study of the mechanisms of negotiation do not fall under the scopes of this paper. Our aim is, instead, to give an effective framework to certify compatibility of transactional aspects between services and invokers.

1.1. *Main contributions*

Our main results are summarised below.

Attribute-based transactions in SOC. We propose a semantics for failure propagation inspired by CMT and adapted to SOC. To this purpose, we introduce a CCS-like process calculus called ATc (after '*A*ttribute-based *T*ransactional *c*alculus') which allows both invokers and callees to specify their own transactional requirements. Our aim is not to provide a semantics for CMT but rather to investigate how CMT could be borrowed to define a rigorous and flexible model for disciplining and analysing the dynamic reconfiguration of transactional scopes in SOC.

Testing framework. We adapt the testing theory (De Nicola and Hennessy 1984) to ATc and propose a formal framework for analysing the interplay between communication failures and the observable behaviour of service-oriented systems. The two parts of the paper relate to each other through *prudent systems*, namely systems that are well-behaved with respect to may-testing (see next paragraph). In some sense, prudent systems reconcile two different classes of failures, namely failures due to service invocations and those due to communication mishaps. Interestingly, prudent systems can be envisaged as a possible way of analysing ATc systems by exploiting the fact that, for prudent systems, may-testing is a pre-congruence for transactional scope contexts (Theorem 6.1).

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Static analysis of distributed transactions. We present a type system to check that no error will occur upon service invocation due to the incompatibility of the transactional scopes of caller and callee (cf. Theorem 4.2). Despite its simplicity, our typing system yields a characterisation of the class of *prudent systems*; intuitively, such systems are well-behaved according to may-testing; in fact, running a process P of a prudent system within a transactional scope does not spoil successful tests, namely $\langle P \rangle$ passes all the tests of P.

Preliminary versions of our results have appeared in (Bocchi and Tuosto 2010a) and (Bocchi and Tuosto 2010b). Here we give a uniform and more complete account with some novel contributions: we extend previous results of our framework proving that – contrary to the general case – some transactional attributes are observationally indistinguishable (cf. Theorem 7.1), and some comparable (cf. Theorem 7.2), under mild conditions. Namely, attributes can be replaced with each other without altering the observable behaviour. Also, under certain conditions some configurations of transactional scope allow comparable observed behaviours (cf. Theorem 6.1). Some practical applications of these results to the design and engineering of business processes over SOAs have been discussed in (Bocchi *et al.* 2010).

1.2. *Synopsis*

The transactional mechanisms of EJB are summarised in Section 2. We present ATc in Section 3 and its typing discipline in Section 4. In Section 5 we present a testing framework for ATc to determine whether ATc systems react correctly to failures. A system is considered correct when it successfully passes some tests (which are defined ad hoc for each process, depending from the application context). In Section 6 we show that in the general case different configurations of transactional scope cause different behaviours (Section 6.1). However for the class of *prudent systems*, the structure of the transactional scopes is preserved upon reduction (Section 6.2). In Section 7 we prove a number of relations on attributes. Section 8 presents conclusions and discusses related work (Section 8.1) and future work (Section 8.2). Proofs are mainly collected in appendices A, B and C.

2. Background: EJB transactional attributes

EJB (EJB 2009; Panda *et al.* 2007) is a middleware for building distributed applications using reusable components. Objects in EJB are executed in specialised run-time environment called *containers*. An EJB container supports typical functionalities to manage e.g., the life-cycle of a *bean* (i.e., a component) and to make components accessible to other components by binding them to a naming service.

EJB containers also offer a mechanism for automatic transaction management, called *container managed transactions* (CMT). CMT relieves the programmer from the burden of explicitly defining the boundaries of each transaction, namely its *transactional scope*, in the code. A transactional scope is a portion of code encompassing a number of actions that constitute a transaction: the failure of an action within a transactional scope causes

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Table 1. *Informal semantics of EJB attributes.*

	Invoker **outside** a scope		Invoker **inside** a scope		Transactional attributes	
	Before invocation	**After** invocation	**Before** invocation	**After** invocation		
(1)	•	• ○	[•]	• ○	`r`	(**Requires**)
(2)	•	• ○	[•]	• [○]	`rn`	(**Requires New**)
(3)	•	• ○	[•]	• ○	`ns`	(**Not Supported**)
(4)	•	⊗	[•]	• ○	`m`	(**Mandatory**)
(5)	•	• ○	[•]	[⊗]	`n`	(**Never**)
(6)	•	• ○	[•]	• ○	`s`	(**Supported**)

the failure of the transaction, and the failure of a transaction affects all the actions in the transactional scope (e.g., they have to be interrupted or rolled back). The container also determines how the transactional scopes should reconfigure upon method invocation. More precisely, a container associates each method of a component with (exactly) one *transactional attribute*: when a method is invoked, the transactional attribute specifies in which transactional scope, if any, the instance of the invoked method should be executed (e.g., in the scope of the action which is invoking the method, outside of any scope, etc.).

EJB features the following transactional attributes:

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

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 Table 1, where transactional scopes are represented by rectangular boxes, the caller is represented by a filled circle • and the callee (or more precisely the new copy of the callee, which is a method in a component) is represented by an empty circle ○. Failed activities are represented by ⊗. Each row represents the behaviour of one transactional attribute and shows how the transactional scope of the caller and the newly created instance of the invoked method behave upon invocation. The first two columns show, respectively, invocations from outside and from within a transactional scope. More precisely, in Table 1:

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 always executed in a new transactional scope;
3. a callee supporting `ns` is always executed outside a transactional scope;
4. the invocation of a method supporting `m` fails if the caller is not in a transactional scope, otherwise the method is executed in the transactional scope of the caller;
5. the invocation of 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);

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

6. a method supporting $\mathtt{s}$ is executed inside (resp. outside) the caller's transactional scope if the caller is executing in (resp. outside) a transactional scope.

2.1. *EJB and ATc: similarities and differences*

As mentioned earlier, the aim of this paper is to define a semantics for reconfiguring transactions over SOAs, not a semantics for EJB. We took inspiration from the mechanisms of automatic handling of reconfiguration of transactional scopes provided by EJB and adapted them to SOAs. Although ATc presents many similarities with EJB, it also differs from the former in a number of aspects.

ATc models the original semantics of each attribute with respect to the reconfiguration of the transactional scopes. However, ATc focuses on *services* rather than *methods*. The main technical difference is that, whereas a method invocation refers to a method of an object offered in a specific container, a service invocation is resolved at run-time discovering, matchmaking and selecting among a number of available implementations offered by possibly different service providers. More precisely:

An invocation can match more than one published service: whereas each method invocation in EJB matches the method of a specific instance, in ATc more than one service implementation may be available, possibly associated with different transactional attributes. Also, the choice among the available services is non-deterministic.

The requester can specify the desired transactional attributes: whereas in EJB the association of one attribute with each method is made only by containers (therefore the invoking party must ensure that the invoked method supports the desired attribute), in ATc both providers and requesters can specify which attribute should be used.

ATc relies on a (service) *environment* which consists of a set of *service providers* (providers for short), each publishing one or more services. We use the environment to abstract the different providers: the environment is a relation that associates

— an abstract service reference s, specifying the offered functional properties,

— one transactional attribute $a \in \mathcal{A}$

to an implementation (e.g., a process). We do not explicitly model the dynamic evolution of the environment caused by the run-time publication and revocation of services by their providers, nor the process of negotiation between requesters and providers. However, we take these aspects into account by assuming that any service invocation eventually matches a suitable service. Namely, an invocation does not fail when a suitable service is not available; rather, it hangs until a service matching the invocation is published.

Unlike in EJB, each service invocation in ATc specifies a set of acceptable transactional attributes, say $A \subseteq \mathcal{A}$. The execution of the invocation triggers, at run-time, the discovery of a service supporting one of the attributes in A. In this case, the invocation of the service triggers a new remote instance of the selected service.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

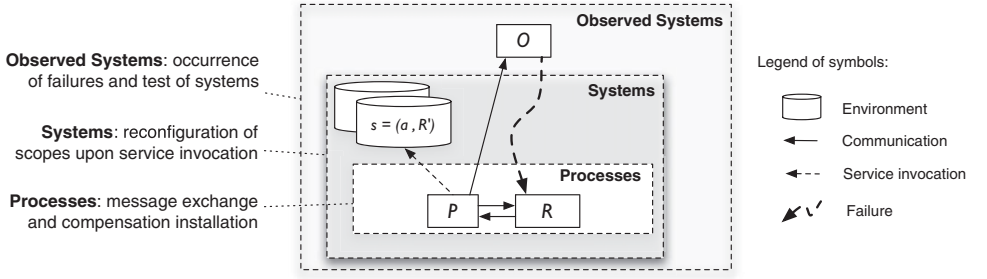


Fig. 1. ATc layered structure.

3. Attribute-based transaction calculus (ATc)

Transactional attributes in Table 1 are embedded in a simple process calculus in order to give a general model for SOC. We define the ATc calculus by taking a technology-agnostic standpoint on the service-oriented paradigm and abstract from its actual realisations.

The ATc calculus is built on top of two layers: *processes* and *systems*; also, systems can run in parallel with observers which can communicate with processes and model communication failures as illustrated in Figure 1.

The layer of *processes* is presented in Section 3.1 and models distributed interactions in presence of possibly nested transactional scopes. When a communication takes place inside a transaction, the compensation associated to that communication is installed in the enclosing transactional scope. The layer of processes does not model failures nor dynamic reconfiguration.

The layer of *systems* is presented in Section 3.2 and yields the formal account for service definition and invocation, as well as for the reconfigurations due to service invocations. Furthermore, systems model those failures that are caused by inappropriate usage of the transactional attributes.

Later (cf. Section 5), we will introduce *observed systems* to study the behaviour of systems in presence of communication failures. The basic idea is that an observer (O in Figure 1) may interact with processes by communicating with them (like O and P in Figure 1), and trigger communication failures (like O and R in Figure 1).

3.1. *ATc processes*

An ATc process is a CCS-like process with three additional capabilities: *service invocation*, *transactional scope*, and *compensation installation* (service definition and invocation will be dealt with in Section 3.2). Let $\mathcal{S}$ and $\mathcal{N}$ be two countably infinite and disjoint sets of names for *services* and *channels*, respectively; we let $s, s', \ldots$ range over $\mathcal{S}$, $x, y, z, \ldots$ range over $\mathcal{N}$, and u range over $\mathcal{S} \cup \mathcal{N}$.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Definition 3.1 (processes). The set $\mathcal{P}$ of *ATc processes* is defined as the set of terms derivable by the following grammar:

$$\begin{array}{lll} P, Q ::= \varnothing & \text{empty process} & \pi ::= x \quad \text{input} \\ \quad \mid \ \nu x\ P & \text{channel restriction} & \quad \mid \ \overline{x} \quad \text{output} \\ \quad \mid \ P \mid Q & \text{parallel} & \\ \quad \mid \ !P & \text{replication} & \\ \quad \mid \ s \propto A.P & \text{service invocation } (A \subseteq \mathcal{A}) & \\ \quad \mid \ \langle P \rangle_Q & \text{transactional scope} & \\ \quad \mid \ \pi \natural Q.P & \text{compensation installation} & \end{array}$$

as usual $\pi = \overline{\overline{\pi}}$ and restriction $\nu x\ P$ binds x in P. The sets of *free* and *bound* channels of $P \in \mathcal{P}$ are respectively denoted by $\mathsf{fc}(P)$ and $\mathsf{bc}(P)$ and, omitting the standard definitions, defined as:

$$\begin{array}{ll} \mathsf{fc}(s \propto A.P) = \mathsf{fc}(P) & \mathsf{bc}(s \propto A.P) = \mathsf{bc}(P) \\ \mathsf{fc}(\langle P \rangle_Q) = \mathsf{fc}(P) \cup \mathsf{fc}(Q) & \mathsf{bc}(\langle P \rangle_Q) = \mathsf{bc}(P) \cup \mathsf{bc}(Q) \\ \mathsf{fc}(x \natural Q.P) = \mathsf{fc}(\overline{x} \natural Q.P) = \mathsf{fc}(P) \cup \mathsf{fc}(Q) \cup \{x\} & \mathsf{bc}(\pi \natural Q.P) = \mathsf{bc}(P) \cup \mathsf{bc}(Q). \end{array}$$

A standard process algebraic syntax is adopted for idle process, restriction, parallel composition and replication. Process $s \propto A.P$ invokes a service s required to support one of the transactional attributes in $A \subseteq \mathcal{A}$ (we abbreviate $s \propto \{a\}.P$ with $s \propto a.P$); a transactional scope $\langle P \rangle Q$ consists of a running process P and a compensation Q assumed to be confined in the transactional scope and executed only upon failure; $\pi \natural Q.P$ executes π and installs the compensation Q in the enclosing transactional scope then behaves as P.

Definition 3.2 (structural congruence). The *structural congruence* $\equiv \subseteq \mathcal{P} \times \mathcal{P}$ is the smallest equivalence relation satisfying the following laws:

$$!P \mid P \equiv !P \qquad \langle \varnothing \rangle_Q \equiv \varnothing \qquad \text{if } P \equiv Q \text{ then } \langle P \rangle_R \equiv \langle Q \rangle_R \text{ and } \langle R \rangle_P \equiv \langle R \rangle_Q$$

$$\nu x\ \langle P \rangle_Q \equiv \langle \nu x\ P \rangle_Q, \text{ if } x \notin \mathsf{fc}(Q) \qquad \nu x\ \langle P \rangle_Q \equiv \langle P \rangle_{\nu x Q}, \text{ if } x \notin \mathsf{fc}(P)$$

$$\nu x\ \nu y\ P \equiv \nu y\ \nu x\ P \qquad \nu x\ \varnothing \equiv \varnothing \qquad \nu x\ (P \mid Q) \equiv (\nu x\ P) \mid Q, \text{ if } x \notin \mathsf{fc}(Q)$$

and closed under α-renaming and the monoidal axioms for $\mid$ and $\varnothing$.

Hereafter, $\pi.P$ stands for $\pi \natural Q.P$ when $Q \equiv \varnothing$ and trailing occurrences of $\varnothing$ are omitted. Also, $\nu x_1, \ldots, x_n\ P$ abbreviates $\nu x_1, \ldots, \nu x_n\ P$.

In ATc, transactional scopes can be nested up to an arbitrary level. The nesting does not alter the communication capabilities of processes; in other terms, transactions do not guarantee isolation and the activities performed within a scope are immediately visible from outside the transactional scope. Transactional scopes influence the behaviour of processes only in case of failure.

To model the semantics of communications we use *contexts*.

Definition 3.3 (contexts). A *context* $\mathbb{C}[\![\square \ ; \boxtimes]\!]$ is either a *scope-avoiding* $\mathbf{A}[\![\square]\!]$ or a *scope-containing* context $\mathbf{C}[\![\square \ ; \boxtimes]\!]$ respectively defined by the following grammars:

$$\mathbf{A}[\![\square]\!] ::= \square \ \mid \ P \mid \square \ \mid \ \square \mid P \qquad \mathbf{C}[\![\square \ ; \boxtimes]\!] ::= \langle \mathbf{C}[\![\square \ ; \boxtimes]\!] \rangle_Q \ \mid \ \langle \square \mid P \rangle_{Q \mid \boxtimes}.$$

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

We often omit the hole for the compensation, e.g., writing $\mathbb{C}[\![\natural]\!]$ when the compensation is not relevant.

Basically, in a scope-avoiding context the hole $\natural$ is not enclosed in a transactional scope while in scope-containing one it is and the hole $\boxtimes$ is always in parallel with compensation of the innermost scope. Definition 3.3 excludes $\nu x\ \mathbb{C}[\![\natural \fatsemi \boxtimes]\!]$ from the set of contexts to avoid name capture. As it will be clear when introducing the transition rules for systems (Definition 3.6), the newly created instance of an invoked service may be executed, for some transactional attributes, in the context of the callee. For example, $\mathbb{C}[\![\langle s \propto \mathtt{r}.P\rangle \fatsemi \boxtimes]\!]$ would reconfigure into $\mathbb{C}[\![\langle P \mid R\rangle \fatsemi \boxtimes]\!]$ (where R is the implementation of service s). Considering, upon reconfiguration, whether the names bound by $\mathbb{C}[\![\natural \fatsemi \boxtimes]\!]$ at the hole are disjoint from $\mathsf{fn}(R)$ would make the semantic more complicated. Also, prefix contexts $\alpha.\mathbb{C}[\![\natural \fatsemi \boxtimes]\!]$ (where α is either of the prefixes of ATc) are ruled out from Definition 3.3 as they prevent inner reductions.

The semantics of ATc process communications is defined by means of the reduction relation in Definition 3.4.

Definition 3.4 (process reduction). The *reduction relation of ATc processes* is the smallest relation $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$ closed under the following axioms and rules:

$$\mathbb{C}[\![\mathbf{C}[\![\pi\sharp Q.P \fatsemi R]\!] \mid \mathbf{C'}[\![\bar{\pi}\sharp Q'.P' \fatsemi R']\!] \fatsemi R_0]\!] \rightarrow \mathbb{C}[\![\mathbf{C}[\![P \fatsemi R \mid Q]\!] \mid \mathbf{C'}[\![P' \fatsemi R' \mid Q']\!] \fatsemi R_0]\!] \quad \texttt{(p1)}$$

$$\mathbb{C}[\![\mathbf{C}[\![\pi\sharp Q.P \fatsemi R]\!] \mid \bar{\pi}\sharp Q'.P' \fatsemi R_0]\!] \rightarrow \mathbb{C}[\![\mathbf{C}[\![P \fatsemi R \mid Q]\!] \mid P' \fatsemi R_0 \mid Q']\!] \quad \texttt{(p2)}$$

$$\mathbf{C}[\![\pi\sharp Q.P \mid \bar{\pi}\sharp Q'.P' \fatsemi R_0]\!] \rightarrow \mathbf{C}[\![P \mid P' \fatsemi R_0 \mid Q \mid Q']\!] \quad \texttt{(p3)}$$

$$\mathbf{A}[\![\pi\sharp Q.P \mid \bar{\pi}\sharp Q'.P']\!] \rightarrow \mathbf{A}[\![P \mid P']\!] \quad \texttt{(p4)}$$

$$\frac{P \rightarrow P'}{\nu x\ P \rightarrow \nu x\ P'} \qquad \frac{P \equiv P' \rightarrow Q' \equiv Q}{P \rightarrow Q}. \quad \texttt{(p5/p6)}$$

Call *transactional* a process enclosed in a transactional scope.

In (p1÷p4) sender and receiver synchronise regardless of the relative nesting of their transactional scopes. If communication actions are executed inside a transactional scope, compensations are installed in the innermost scope enclosing the communicating process, otherwise they are discarded. In (p1) the communication is between two parallel transactional processes in different transactional scopes. In (p2) the communication is between a transactional process and a second process which is transactional or not (in the former case, the transactional scope includes the one of the first process). In (p3) the communication is between two transactional processes included in the same innermost transactional scope. In rule (p4) the communication is between two non-transactional processes. Rule (p5) is necessary since contexts do not include hiding while (p6) is the standard structural congruence rule.

Remark 3.1. Rules (p3) and (p4) could be merged in the rule

$$\mathbb{C}[\![\pi\sharp Q.P \mid \bar{\pi}\sharp Q'.P' \fatsemi R_0]\!] \quad \rightarrow \quad \mathbb{C}[\![P \mid P' \fatsemi Q \mid Q' \mid R_0]\!].$$

For clarity, we prefer the presentation of Definition 3.4.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Like in (Guidi *et al.* 2009), Definition 3.4 allows for dynamic installation of compensations that, in our case, are installed in parallel with existing ones (whereas in (Guidi *et al.* 2009) it is possible to specify any kind of composition). Also, only the actions executed before the failure are compensated, as illustrated by Example 3.1.

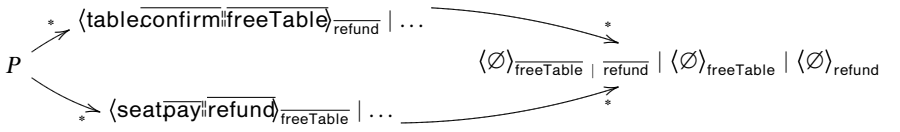
Example 3.1. Consider the transactional scope $P_{\mathsf{bookNight}} = \langle P_{\mathsf{theatre}} \mid P_{\mathsf{dinner}} \rangle$ where:

$$P_{\mathsf{theatre}} = \overline{\mathsf{seat}}.\ \mathsf{seat}.\ \overline{\mathsf{pay}} \| \overline{\mathsf{refund}} \qquad P_{\mathsf{dinner}} = \overline{\mathsf{table}}.\ \mathsf{table}.\ \overline{\mathsf{confirm}} \| \overline{\mathsf{freeTable}}$$

where $\overline{\mathsf{refund}}$ and $\overline{\mathsf{freeTable}}$ are dynamically installed to compensate $\overline{\mathsf{pay}}$ and $\overline{\mathsf{confirm}}$, respectively. Assume that $P_{\mathsf{bookNight}}$ runs in parallel with two servers $\langle Q_{\mathsf{theatre}} \rangle$ and $\langle Q_{\mathsf{dinner}} \rangle$ for theatre and restaurant booking defined as

$$Q_{\mathsf{theatre}} = \mathsf{seat}.\ \overline{\mathsf{seat}}.\ \mathsf{pay} \| \mathsf{refund} \qquad Q_{\mathsf{dinner}} = \mathsf{table}.\ \overline{\mathsf{table}}.\ \mathsf{confirm} \| \mathsf{freeTable}$$

According to Definition 3.4 the process $P = \langle P_{\mathsf{theatre}} \mid P_{\mathsf{dinner}} \rangle \mid \langle Q_{\mathsf{theatre}} \rangle \mid \langle Q_{\mathsf{dinner}} \rangle$ has, among others, the following two executions:



Notice that a different interleaving of the interactions would let the client execute different compensations in case of failure. ◇

3.2. *ATc systems*

The semantics of service invocations is given at the level of *systems* (Definition 3.5) where processes can invoke services published in the environment. An ATc environment models a distributed service repository where a service can be published by one or more providers. More precisely, an environment is defined as a relation $\Gamma \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{P}$; if $(s, a, P) \in \Gamma$, then the service s has 'body' P and supports the attribute a. An invocation of s creates a new instance of the service that executes P. A service s can be associated to more than one pair in $\mathcal{A} \times \mathcal{P}$ since more than one provider may publish an implementation for s (and possibly associate it to different transactional attributes).

Definition 3.5 (systems). A *system* in ATc is a pair $\Gamma \vdash P$ where P is derived by the productions in Definition 3.1 augmented with $P ::= \mathtt{err}$ to represent *erroneous processes*. Also, the following axioms

$$!\mathtt{err} \equiv \mathtt{err} \qquad \nu x\ \mathtt{err} \equiv \mathtt{err} \qquad \langle \mathtt{err} \rangle_Q \equiv \mathtt{err}$$

extend the congruence relation (cf. Definition 3.2) to erroneous processes.

Given $A \subseteq \mathcal{A}$, hereafter we adopt the following conventions:

— $P \in \Gamma(s, A)$ shortens $\exists a \in A \ :\ (s, a, P) \in \Gamma$ and models a provider of s implemented as P and supporting a (we shorten $P \in \Gamma(s, \{a\})$ as $P \in \Gamma(s, a)$);
— $S, S', \ldots$ range over systems and $P, Q, \ldots$ range over both $\mathcal{P}$ and erroneous processes (note that processes in $\mathcal{P}$ are non-erroneous);

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

— terms where compensations contain `err` are ruled out; 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. it is not) executed inside a transactional scope.

Definition 3.6 formalises the CMT mechanisms which are rendered in SOC by allowing environments Γ 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. We write $\tilde{x}$ for the tuple of names $x_1, \ldots, x_n$.

Definition 3.6 (systems' reduction). The *reduction relation of ATc systems* is the smallest relation $\rightsquigarrow$ closed under the rules/axioms below:

$$\frac{P \to P'}{\Gamma \vdash P \rightsquigarrow \Gamma \vdash P'} \quad \texttt{(s1)}$$

$$\frac{\mathtt{m} \in A}{\Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![s \propto A.P]\!] \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![\mathtt{err}]\!]} \quad \texttt{(s2)}$$

$$\frac{R \in \Gamma(s, \{\mathtt{s}, \mathtt{n}, \mathtt{ns}\} \cap A)}{\Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![s \propto A.P]\!] \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ (\mathbf{A}[\![P]\!]) \mid R} \quad \texttt{(s3)}$$

$$\frac{R \in \Gamma(s, \{\mathtt{r}, \mathtt{rn}\} \cap A)}{\Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![s \propto A.P]\!] \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ (\mathbf{A}[\![P]\!]) \mid \langle R \rangle} \quad \texttt{(s4)}$$

$$\frac{\tilde{x} \cap \mathsf{fc}(R) = \varnothing \quad R \in \Gamma(s, \{\mathtt{m}, \mathtt{s}, \mathtt{r}\} \cap A)}{\Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\mathbf{C}[\![s \propto A.P \fatsemi Q]\!] \fatsemi Q']\!] \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\mathbf{C}[\![P \mid R \fatsemi Q]\!] \fatsemi Q']\!]} \quad \texttt{(s5)}$$

$$\frac{\mathtt{n} \in A}{\Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\langle s \propto A.P \mid P' \rangle_Q \fatsemi Q']\!] \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![Q \fatsemi Q']\!]} \quad \texttt{(s6)}$$

$$\frac{\mathtt{ns} \in A \qquad R \in \Gamma(s, \mathtt{ns})}{\Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\mathbf{C}[\![s \propto A.P \fatsemi Q]\!] \fatsemi Q']\!] \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ (\mathbb{C}[\![\mathbf{C}[\![P \fatsemi Q]\!] \fatsemi Q']\!]) \mid R} \quad \texttt{(s7)}$$

$$\frac{\mathtt{rn} \in A \qquad R \in \Gamma(s, \mathtt{rn})}{\Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\mathbf{C}[\![s \propto A.P \fatsemi Q]\!] \fatsemi Q']\!] \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ (\mathbb{C}[\![\mathbf{C}[\![P \fatsemi Q]\!] \fatsemi Q']\!]) \mid \langle R \rangle}. \quad \texttt{(s8)}$$

Rule `(s1)` lifts process behaviour to systems while rules `(s2÷s8)` yield the semantics of the transactional attributes informally presented in Section 2. The presence of restrictions in rules `(s2÷s8)` takes into account name scoping; the reductions are possible because of the structural laws on processes. Rules `(s2÷s4)` model the first column of Table 1 (i.e., non-transactional invocations) and `(s5÷s8)` model the second one (i.e., transactional invocations); below we comment on such rules.

Rule `(s2)` states that a non-transactional invocation of a service supporting attribute `m` results in a failure. Rule `(s3)` states that a non-transactional invocation of a service supporting either `s`, `n`, or `ns`, causes the execution of the new service instance in parallel with the continuation of the caller. By rule `(s4)`, a service supporting `r` or `rn` will be executed in a new scope (initially with idle compensation). By rule `(s5)`, in case of transactional invocation of a service supporting attributes `m`, `s`, or `r`, the service instance is

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

executed in the same scope of the caller. The channel names $\tilde{x}$ should not appear free in R since the service instance is executed within the context of the invoking process, which is within the scope of $\nu\tilde{x}$. Rule (s6) states that a transactional invocation of a service supporting n results in a failure. Notice that, whereas failures occurring outside a scope make the process to become erroneous, i.e., in (s2), failures occurring inside a scope trigger its compensation, i.e., in (s6). By rule (s7), a transactional invocation requesting ns will let the service instance to run outside the caller's scope. Finally, rule (s8) states that a transactional invocation requesting rn will let the service instance to run in a new scope with idle compensation.

Remark 3.2. Rule (s2) may appear too restrictive as it introduces an error even if Γ may offer a service supporting other attributes in A. An actual implementation may in fact select more suitable services if there are any available. Here, more simply, we define the conditions to correctly use attributes avoiding errors in *any* possible environment; therefore (s2) models the worst case scenario.

The following examples motivate the need of a disciplined use of transactional attributes. The typing system presented in Section 4, ensures that a well-typed process will not produce errors due to the fact that the attributes required by an invoker are not suitable for the configuration of the transactional scope of the invoking process.

Example 3.2. Let $\langle s_{\mathsf{tickets}} \propto \mathtt{m}.P_{\mathsf{theatre}} \rangle_{s_{\mathsf{compensate}} \propto \mathtt{m}}$ be a process that invokes s_{tickets} and behaves as P_{theatre}. Noticeably, in P_{theatre} a failure cannot occur by rule (s6) since P_{theatre} is executed inside a transactional scope. Nevertheless, as it will be clearer from Section 5, the interference of an observer may cause a communication failure (see Definition 5.2). If such a failure occurs in a communication with P_{theatre} then the compensation $s_{\mathsf{compensate}} \propto \mathtt{m}$ is executed outside a transactional scope. Therefore, the non-transactional invocation to $s_{\mathsf{compensate}}$ will result in an error. ◇

Example 3.3. Let $P_{\mathsf{theatre}} = \overline{\mathsf{askSeat}}.\mathsf{getSeat}\overline{\mathsf{pay}} \,|\, \overline{\mathsf{getRefund}}$ and consider

$$P_{\mathsf{bookTheatre}} = s_{\mathsf{tickets}} \propto \{\mathtt{s}, \mathtt{n}, \mathtt{ns}, \mathtt{r}, \mathtt{rn}\}.P_{\mathsf{theatre}}$$

$$P_{\mathsf{tickets}} \quad = \mathsf{askSeats}.\overline{\mathsf{getSeats}}.s_{\mathsf{bank}} \propto \mathtt{m}$$

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

On the one hand, requesters should check that the attributes of their service calls are compatible with the context (transactional or not) in which the calls are done. On the other hand, providers must guarantee that none of their services yields errors; namely, the execution of (the body of) a service in any context resulting from its supported attributes should be safe. For instance, s_{tickets} in Example 3.3 supports s by which the new service instance can be executed either inside or outside a transactional scope, depending on whether the invocation is done from inside or outside a transactional scope. Therefore, the execution P_{tickets} should be safe regardless it will run inside or outside a transactional scope.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

4. A type system for transactional services

This section yields a type system for ATc that can detect possible failures of invocations due to misuse of transactional attributes. We give an algebra of types (Section 4.1), then define a type system for ATc (Section 4.2), and finally we give a suitable notion of well-typedness for ATc systems (Section 4.3) which is preserved by the reduction relation (Theorem 4.2) and ensures error-freedom (Theorem 4.3).

4.1. *Types for ATc*

Our types record, for each possible invocation, its transactional modality (i.e., if it is transactional or not) and the set of transactional attributes associated with it.

Definition 4.1 (types). 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 follows:

$$t ::= \varnothing \mid (I, t, t) \qquad (Types).$$

We write $P \triangleright t$ to state that $P \in \mathcal{P}$ has type t.

If $P \triangleright \varnothing$ 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 yields 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 processes pending in prefixes of P possibly outside any transactional scope.

Remark 4.1. The invocations in installation prefixes t_u have to be considered only if the process is within a transactional scope. In fact, installations associated with synchronisations occurring outside transactional scopes vanish by Definition 3.6. Invocations in t_c are transactional only if the transactional scope is nested inside another transactional scope. For example, upon failure, $\mathbb{C}[\![\langle P \rangle_Q \fatsemi R]\!]$ moves to $\mathbb{C}[\![Q \fatsemi R]\!]$, thus the invocations in the compensation Q are (resp. are not) transactional if $\mathbb{C}[\![\natural \fatsemi \boxtimes]\!]$ is (resp. is not) transactional.

Examples 4.1 and 4.2 give an intuitive illustration of typing in ATc by showing two simple processes and their respective types. The formal definition of the typing rules is presented in Section 4.2.

Example 4.1. Consider $P_2 = s \propto A y \| P_1$ with $P_1 \triangleright t_1$. The type of P_2 is defined below:

$$P_2 \triangleright t_2 \quad \text{with} \quad t_2 = (\{\mathfrak{o}\} \times A, \varnothing, t_1)$$

The first component of t_2 collects information on the invocation of P_2 to s, which is non-transactional and with attributes A. The third component of t_2 is t_1 which is the type of the process P_1 to be installed as a compensation for prefix y. ◇

Example 4.2. Take the process $P_3 = \langle P_2 \rangle$, where P_2 is defined in Example 4.1,

$$P_3 \triangleright t_3 \quad \text{with} \quad t_3 = (\{\mathfrak{i}\} \times A, t_1, \varnothing).$$

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

The first component of t_3 records the invocations of P_2 (transactional, in this case); therefore, the first component of t_3 is $\{\mathrm{i}\} \times A$. Since P_2 is inside a transaction the compensations in t_1 are installed (thereby in the second component of t_3). $\diamond$

The next example illustrates the use of the second components of types.

Example 4.3. Let $P_4 = \langle P_3 \rangle_{P_C}$ where P_3 (cf. Example 4.2) is in a transactional scope with compensation P_C. Assume $P_C \triangleright t_c$; then

$$P_4 \triangleright t_4 \qquad \text{with} \quad t_4 = (\{\mathrm{i}\} \times (A \cup A_1), t_c, \varnothing)$$

where $t_1 = (\{\mathrm{o}\} \times A_1, \varnothing, \varnothing)$. Observe that the invocations in the second component of t_3 are now merged in the first component of t_4 (where they appear to be transactional). $\diamond$

Note that in Example 4.3 the second and third components of t_1 could be non-idle; this would make the 'merging' of t_1 and t_4 more complex (the rules for the general case are presented in Definition 4.2).

It is convenient to treat types as binary trees whose nodes are labelled with subsets of $\{\mathrm{i}, \mathrm{o}\} \times \mathcal{A}$. 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 ($\varnothing$ is the empty tree which is conventionally labelled with the empty set). The operators $_\downarrow_1$, $_\downarrow_2$, and $_\downarrow_3$ are used to 'traverse' types (and respectively are the projections of the tuple-like representation of types); the operator $_ \oplus _$ 'adds' types; the formal definitions of such operators are:

$$\varnothing\downarrow_1 = \varnothing, \quad \varnothing\downarrow_2 = \varnothing\downarrow_3 = \varnothing, \quad (I, t_c, t_u)\downarrow_1 = I, \quad (I, t_c, t_u)\downarrow_2 = t_c, \quad (I, t_c, t_u)\downarrow_3 = t_u$$

$$\varnothing \oplus t = t, \quad (I, t_c, t_u) \oplus (I', t'_c, t'_u) = (I \cup I', t_c \oplus t'_c, t_u \oplus t'_u)$$

($_ \oplus _$ has lower precedence than unary operators). We identify $(\varnothing, \varnothing, \varnothing)$ and $\varnothing$ because they have the same behaviour with respect to the type operators.

4.2. *Typing ATc*

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

Definition 4.2 (typing rules). The typing rules for non-erroneous processes (cf. Definition 3.5) are

$$\texttt{(idle)}\ \frac{}{\varnothing \triangleright \varnothing} \qquad \frac{P \triangleright t}{\nu x\, P \triangleright t}\ \texttt{(res)}$$

$$\texttt{(par)}\ \frac{P \triangleright t \qquad P' \triangleright t'}{P \mid P' \triangleright t \oplus t'} \qquad \frac{P \triangleright t}{!P \triangleright t}\ \texttt{(repl)}$$

$$\texttt{(inv)}\ \frac{P \triangleright t_p \qquad I = \{\mathrm{o}\} \times A}{s \propto A.P \triangleright (I \cup t_p\downarrow_1, t_p\downarrow_2, t_p\downarrow_3)} \qquad \frac{P \triangleright t_p \qquad Q \triangleright t_q}{\pi \,\|\, QP \triangleright (t_p\downarrow_1, t_p\downarrow_2, t_q \oplus t_p\downarrow_3)}\ \texttt{(comp)}$$

$$\texttt{(scope)}\ \frac{P \triangleright t \qquad I = t\downarrow_1 \qquad t_c = t\downarrow_2 \qquad t_u = t\downarrow_3 \qquad Q \triangleright t_q}{\langle P \rangle_Q \triangleright ((I \cup t_c\downarrow_1)[\![\mathrm{o} \mapsto \mathrm{i}]\!], t_u \oplus t_c\downarrow_2 \oplus t_c\downarrow_3 \oplus t_q, \varnothing)}$$

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

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

The first five rules in Definition 4.2 are straightforward. Rule (comp) states that the type of the installation of a compensation Q records the invocations in Q as potential invocations of P by adding them to the third component of the type of $\pi \Vdash Q.P$. Rule (scope) is the most complex rule therefore we give a more detailed explanation. Recall that, given a type (I, t_c, t_u), t_c is the type of the compensations installed in the main process and t_u is the type of the compensations to install. The information in t_c is kept separated from I until (in the proof tree) it is possible to determine whether the compensations are enclosed in a transactional scope (i.e., the main process is enclosed in at least two transactional scopes). The information of t_u is kept separated until it is possible to determine whether the respective compensations will be actually installed in an enclosing transactional scope or discarded. In (scope), P is enclosed in a transactional scope therefore the compensations of P will be surely executed inside a transactional scope (as in Example 4.3). Hence, if t' denotes the type of $\langle P \rangle_Q$ in the conclusion of the rule (scope)

— the invocations of P (i.e., $t\downarrow_1$) and its installed compensation (i.e., $t_c\downarrow_1$) are included in the first component of t' applying the substitution $[\![\mathfrak{o} \mapsto \mathfrak{i}]\!]$;
— the other components of t_c (i.e., $t_c\downarrow_2$ and $t_c\downarrow_3$) are added to those in the compensation Q and to those that prefixes in P will install (i.e., t_u); in fact, the latter invocations become possible, hence they are recorded in $t'\downarrow_2$ in (scope);
— finally, $t'\downarrow_3 = \emptyset$ since the pending compensations of prefixes of P would be installed in parallel with Q.

Example 4.4 illustrates how (scope) acts.

Example 4.4. Let us type $\langle P \rangle$ where $P = \langle s \propto \mathtt{m} \rangle_{\pi \Vdash s' \propto \mathtt{m}}$. First we have

$$\frac{s \propto \mathtt{m} \rhd ((\mathfrak{o},\mathtt{s}),\emptyset,\emptyset) \qquad \pi \Vdash s' \propto \mathtt{m} \rhd (\emptyset,\emptyset,(\mathfrak{o},\mathtt{m}))}{\langle s \propto \mathtt{s} \rangle_{\pi \Vdash s' \propto \mathtt{m}} \rhd ((\mathtt{i},\mathtt{s}),(\emptyset,\emptyset,(\mathfrak{o},\mathtt{m})),\emptyset)} \ \texttt{(scope)}$$

namely, the type of the compensation is held in the second component. Hence, again by rule (scope), $\langle \langle s \propto \mathtt{s} \rangle_{\pi \Vdash s' \propto \mathtt{m}} \rangle \rhd ((\mathtt{i},\mathtt{s}),((\mathfrak{o},\mathtt{m}),\emptyset,\emptyset),\emptyset)$. ◇

In Example 4.4, the installations of the compensations will not be discarded but installed in the transactional scope enclosing P. Therefore, the invocation in the installation $(\mathfrak{o},\mathtt{m})$ is moved to the first component of the type of the second component. Note that the type of $\langle\langle P \rangle\rangle$ (where P is the process in Example 4.4) is $(((\mathtt{i},\mathtt{s}),(\mathtt{i},\mathtt{m})),\emptyset,\emptyset)$ and it is obtained by applying again (scope). Indeed, the invocation of s' will be executed inside the outermost transactional scope, hence (scope) adds $(\mathtt{i},\mathtt{m})$ to the first component of the resulting type.

The following example illustrates the typing of a more complex process.

Example 4.5. Consider the process $P = \pi_1 \Vdash Q$ where

$$Q = \pi_2 \Vdash R \qquad \text{and} \qquad R = s_1 \propto A_1 \pi_3 \Vdash s_2 \propto A_2$$

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

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

$$\dfrac{\dfrac{\dfrac{\dfrac{I_2 = \{\mathfrak{o}\} \times A_2 \qquad \emptyset \triangleright \emptyset}{s_2 \propto A_2 \triangleright (I_2, \emptyset, \emptyset)} \qquad \emptyset \triangleright \emptyset}{\pi_3 \| s_2 \propto A_2 \triangleright (\emptyset, \emptyset, I_2)} \qquad I_1 = \{\mathfrak{o}\} \times A_1}{s_1 \propto A_1 \pi_3 \| s_2 \propto A_2 \triangleright (I_1, \emptyset, I_2)} \qquad \emptyset \triangleright \emptyset}{\pi_2 \| R \triangleright (\emptyset, \emptyset, (I_1, \emptyset, I_2))} \qquad \emptyset \triangleright \emptyset}{\pi_1 \| Q \triangleright (\emptyset, \emptyset, (\emptyset, \emptyset, (I_1, \emptyset, I_2)))}$$

(inv) (inv) (Comp) (Comp) (Comp)

◇

Proposition 4.1. For each non-erroneous $P \in \mathcal{P}$

— there is a unique type t such that $P \triangleright t$, and
— for all non-erroneous $Q \in \mathcal{P}$, if $P \equiv Q$ and $P \triangleright t$ then $Q \triangleright t$

Proof. Straightforward by induction on the structure of P and the derivation of the structural congruence. □

The tree-like structure of ATc types is useful to apply the rules in Definition 4.2. Once a process P is assigned its type t, we need to extract from t the actual set of invocations done by P. To this aim we define flat types below.

Definition 4.3. The *flat type* $\widehat{t}$ of t is defined as follows:

$$\widehat{\emptyset} = \emptyset \qquad \widehat{t} = t\downarrow_1 \cup \textsc{Flatten}(t\downarrow_2), \text{ if } t \neq \emptyset$$

where

$$\textsc{Flatten}(\emptyset) = \emptyset \qquad \textsc{Flatten}(t) = t\downarrow_1 \cup \textsc{Flatten}(t\downarrow_2) \cup \textsc{Flatten}(t\downarrow_3), \text{ if } t \neq \emptyset$$

is a function that 'flattens' types into sets.

In the interpretation of a type t as a tree, the flat type of t is the union of the sets labelling all the nodes of t, excluding those of the subtree $t\downarrow_3$ which corresponds to 'dead code' (cf. Example 4.6); if the typed process is not in a scope then its pending compensations can be ignored, otherwise $t\downarrow_3$ is empty because of rule `(scope)`.

Lemma 4.1. For any types t and t', $\widehat{t \oplus t'} = \widehat{t} \cup \widehat{t'}$.

Proof. Directly from the definition of $_ \oplus _$ and from an auxiliary lemma (i.e., that for any types t and t', $\textsc{Flatten}(t \oplus t') = \textsc{Flatten}(t) \cup \textsc{Flatten}(t')$) whose proof, relegated to appendix A.1, is by induction on the structure of $t \oplus t'$. □

4.3. *Well-typedness in ATc*

The definition of well-typedness requires some care. We must adopt a different notion of well-typedness depending on whether an ATc process is intended to be published as a service or not (e.g., run by a user in the traditional way, without run-time discovery). In Definitions 4.4 and 4.5 below we implicitly use Proposition 4.1 that guarantees that non-erroneous processes have a unique corresponding type.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

If P is not published as a service then it suffices to specify, for each service invocation in P, the attributes for which no run-time errors are possible. This enables us to adopt the following definition.

Definition 4.4 (well-typedness for processes). A process $P \in \mathcal{P}$; is *well-typed* iff $(\mathtt{o},\mathtt{m}) \notin \widehat{t}$, where $P \triangleright t$.

Example 4.6. Process P in Example 4.5 is (trivially) well-typed since $\widehat{t} = \varnothing$. 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). ◇

Intuitively, a system is correct if it executes without producing erroneous processes. The correctness of systems depends not only on the well-typedness of the processes that are in the initial configuration of the system, but also on the well-typedness of the services invoked by those processes.

Ensuring correctness for services is a bit more complex than ensuring it for simple processes, since a service instance may be executed in different (transactional) contexts. Whether or not the invocations in the body of (the instance of) a service, say s, are transactional depends on which attributes s supports and on whether the invocation to s is transactional or not; well-typedness of services must consider both the possibilities according to the attributes s has to support.

Definition 4.5 (well-typedness for services). Let $(s,a,P) \in \mathcal{S} \times \mathcal{A} \times \mathcal{P}$. Service s is *well-typed in* (s,a,P), if P is non-erroneous and both (4) and (5) below hold.

$$\langle P \rangle \triangleright t \wedge a \in \{\mathtt{r},\mathtt{rn},\mathtt{m},\mathtt{s}\} \implies (\mathtt{o},\mathtt{m}) \notin \widehat{t} \tag{4}$$

$$P \triangleright t \wedge a \in \{\mathtt{s},\mathtt{n},\mathtt{ns}\} \implies (\mathtt{o},\mathtt{m}) \notin \widehat{t}. \tag{5}$$

An environment Γ is *well-typed* iff for all $(s,a,P) \in \Gamma$, s is well-typed in (s,a,P).

Example 4.7. Let the process P in Example 4.5 be 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 4.6, P is well-typed. For $\langle P \rangle$ we just need to apply rule (scope) as follows:

$$\frac{\pi_1 \| Q \triangleright (\varnothing,\varnothing,(\varnothing,\varnothing,(I_1,\varnothing,I_2))) \qquad \varnothing \triangleright \varnothing}{\langle \pi_1 \| Q \rangle_{\varnothing} \triangleright (\varnothing,(\varnothing,\varnothing,(I_1,\varnothing,I_2)),\varnothing)}\ (\mathtt{scope}).$$

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

Our notion of well-typedness is stricter than necessary; a weaker notion could be adopted by defining flat types that contain the labels of only some of the 'right children'. Though yielding less restrictive types, this would make the definition of flat types (Definition 4.3) more complex and less clear (we opted for simplicity rather than generality).

Proposition 4.2. For any context $\mathbb{C}[\![\square \fatsemi \boxtimes]\!]$ and $\pi \| Q.P, R, R_0 \in \mathcal{P}$, if $\mathbb{C}[\![\langle \pi \| Q.P \rangle_R \fatsemi R_0]\!] \triangleright t$ then $\mathbb{C}[\![\langle P \rangle_{R|Q} \fatsemi R_0]\!] \triangleright t$.

Proof. The proof is by induction on the structure of $\mathbb{C}[\![\square \fatsemi \boxtimes]\!]$ (see appendix A.2). □

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Proposition 4.3. For any scope-avoiding context $\mathbf{A}[\![\natural]\!]$ and $\pi\sharp QP, R \in \mathcal{P}$,

$$\mathbf{A}[\![\pi\sharp QP]\!] \triangleright t \wedge \mathbf{A}[\![P]\!] \triangleright t' \implies t = t' \oplus (\varnothing, \varnothing, t_Q)$$

where $Q \triangleright t_Q$.

Proof. Let $P \triangleright t_P$. If $\mathbf{A}[\![\natural]\!] = \natural$ then $t = (t_P\downarrow_1, t_P\downarrow_2, t_Q \oplus t_P\downarrow_3)$ by direct application of the hypotheses. In the other cases the proof is similar by using rule (par). □

The following theorem is a straightforward implication of Propositions 4.2 and 4.3.

Theorem 4.1. For any $P, Q \in \mathcal{P}$, if P is well-typed and $P \to Q$ then Q is well-typed.

Proof. For any $P, Q \in \mathcal{P}$, let $P \triangleright t_P$ and $Q \triangleright t_Q$, if $P \to Q$ then $\widehat{t_Q} \subseteq \widehat{t_P}$ by Propositions 4.2 and 4.3. □

Theorem 4.2. Let the environment Γ and $P \in \mathcal{P}$ be well-typed. If $\Gamma \vdash P \rightsquigarrow \Gamma \vdash Q$ then Q is well-typed.

Proof. The proof is by case analysis on the derivation of $\Gamma \vdash P \rightsquigarrow \Gamma \vdash Q$ (cf. Definition 3.6), and is relegated in appendix A.3. □

A straightforward corollary of Theorem 4.2 is

Theorem 4.3. If Γ and $P \in \mathcal{P}$ are well-typed and $\Gamma \vdash P \rightsquigarrow \Gamma \vdash Q$ then Q is a non-erroneous process.

5. Testing ATc systems

The intuition behind the testing theory (De Nicola and Hennessy 1984) is that an *observer* interacts with the system to check whether it passes or not a test; two systems are equivalent if they pass the same tests. In this section, we tailor the testing theory to ATc. After motivating our approach in Section 5.1, we define the class of observers used to model communication failures in Section 5.2. Well-typedness is linked to observed systems in Section 5.3 where Theorems 5.1 and 5.2 show that well-typed systems are free from communication errors. Then, Section 5.4 gives an observational semantics relying on the notion of observer defined in Section 5.2.

5.1. *A motivating example*

Intuitively, ATc systems are tested by defining a set of *observers* that reveal how systems react to communication failures. Two systems are considered equivalent when they react in the same way to the same set of observers.

Before introducing the testing framework for ATc, we give an example that highlights the interplay of transactional scopes reconfiguration with the observed behaviour of systems. The example motivates the designed choices of ATc by showing that two systems differing only for one transactional attribute may behave differently.

Consider a scenario, where a service s acts as a proxy R of a shared resource. The resource is not explicitly represented and can be thought of as a process running in

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

parallel with R, that accepts an input on p (to acquire the resource) and an input on v (to release it). The body R of s is

$$R \;=\; \overline{\mathsf{p}}\,\overline{\mathsf{v}}\mathsf{u}.\mathsf{q}.\overline{\mathsf{v}}$$

The expected behaviour is that, upon invocation, s interacts with the resource to acquire a lock on the resource through the action $\overline{\mathsf{p}}\,\overline{\mathsf{v}}$; the compensation $\overline{\mathsf{v}}$ is meant to release the resource if a failure occurs during the access. Then, s waits for a request of usage from the client (action u). When the client does not need the resource any longer, it stops the computations by sending a signal (on q) to s that finally releases the resource ($\overline{\mathsf{v}}$).

Let Γ be an environment such that $R \in \Gamma(\mathtt{m}, s)$ 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. Both providers enforce R to be executed within a transactional scope, so that the compensation $\overline{\mathsf{v}}$ of $\overline{\mathsf{p}}$ can be installed.

Consider the two possible clients P_1 and P_2 below:

$$P_1 = \langle s \propto \mathtt{m}.\overline{\mathsf{u}}.\overline{\mathsf{q}} \rangle \qquad\qquad P_2 = \langle s \propto \mathtt{rn}.\overline{\mathsf{u}}.\overline{\mathsf{q}} \rangle$$

where P_2 is obtained by replacing the attribute $\mathtt{m}$ with $\mathtt{rn}$ in the invocation to s of P_1. Both clients invoke s, use ($\overline{\mathsf{u}}$), and then release the resource ($\overline{\mathsf{q}}$). The different attributes generate two different behaviours as the invocations of s from P_1 and P_2 result in the following systems (by rules $\mathtt{(s5)}$ and $\mathtt{(s8)}$ in Definition 3.6, respectively)

$$S_1 = \Gamma \vdash \langle \overline{\mathsf{u}}.\overline{\mathsf{q}} \mid \overline{\mathsf{p}}\,\overline{\mathsf{v}}\mathsf{u}.\mathsf{q}.\overline{\mathsf{v}} \rangle \qquad\qquad S_2 = \Gamma \vdash \langle \overline{\mathsf{u}}.\overline{\mathsf{q}} \rangle \mid \langle \overline{\mathsf{p}}\,\overline{\mathsf{v}}\mathsf{u}.\mathsf{q}.\overline{\mathsf{v}} \rangle$$

Observe the difference between S_1 and S_2; the invocation of s activates an instance of R in both cases, however in S_1 the new instance of the service runs in the same transactional scope of the invoker (due to the attribute $\mathtt{m}$), while in S_2 it runs in a different transactional scope (due to the attribute $\mathtt{rn}$).

Now take the following process $O = \mathsf{p}.\lightning\overline{\mathsf{u}}.\mathsf{v}.\checkmark$, where $\lightning\overline{\mathsf{u}}$ is an action that forces a failure of $\overline{\mathsf{u}}$ and $\checkmark$ is a distinguished action to denote the success of the test; in other words, O is an *observer* that checks if, after the resource is acquired, it is released regardless of possible failures on clients' requests of use. The executions of S_1 and S_2 in parallel with O result, after the synchronisation on p, in the continuation of O $\lightning\overline{\mathsf{u}}.\mathsf{v}.\checkmark$ to run in parallel with either of the following systems:

$$S_1' = \Gamma \vdash \langle \overline{\mathsf{u}}.\overline{\mathsf{q}} \mid \mathsf{u}.\mathsf{q}.\overline{\mathsf{v}} \rangle_{\overline{\mathsf{v}}} \qquad\qquad S_2' = \Gamma \vdash \langle \overline{\mathsf{u}}.\overline{\mathsf{q}} \rangle \mid \langle \mathsf{u}.\mathsf{q}.\overline{\mathsf{v}} \rangle_{\overline{\mathsf{v}}}$$

Intuitively, O can tell apart S_1' and S_2'. In fact, in S_1' the failure $\lightning\overline{\mathsf{u}}$ forced by O triggers the compensation $\overline{\mathsf{v}}$ which eventually releases the resource so that O ends with the success action, while in system S_2' this is not the case as O is stuck after forcing the failure.

In general, different attributes generate different observational behaviours. Interestingly, as shown in Section 7, it is sometimes possible to inter-change the transactional attributes while preserving the observed behaviour of a system.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

5.2. *Observed systems*

The notion of *observer* is pivotal in our setting. Observers abstract a faulty communication infrastructure and are formally defined as follows.

Definition 5.1 (observers). An *observer* is derived by the following grammar:

$$\begin{array}{lll} O ::= \emptyset & \text{empty process} & \mid\ O + O \quad \text{sum} \\ \mid\ \checkmark & \text{success} & \mid\ \mathbf{rec}\ X.O \quad \text{recursion} \\ \mid\ \pi.O & \text{prefix} & \mid\ X \quad \text{variable.} \\ \mid\ \lightning\pi.O & \text{failure} & \end{array}$$

The structural congruence is extended with the monoidal axioms of $+$.

The set Obs of *observed systems* (called *states*) is the set of pairs of the form $\Gamma \vdash P \parallel O$ where $\Gamma \vdash P$ is a system. Hereafter, when writing $(\Gamma \vdash P) \parallel O$ we mean the observed system $\Gamma \vdash P \parallel O$.

Basically, observers are 'low-level' processes modelling some aspects of the communication environment where ATc processes run. Observers can be thought of as the transport layer where communication faults may happen. Failing and successful tests are represented by $\emptyset$ and $\checkmark$, respectively; prefix $\pi.O$ allows observers to communicate with the system, while prefix $\lightning\pi.O$ causes the failure of a communication capability π 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).

Example 5.1. The observer in Section 5.1 ($O = \mathsf{p}.\lightning\overline{\mathsf{u}}.\mathsf{v}.\checkmark$) represents a test that checks if the system will release the resource in case of failure but it sets no requirements on the normal execution. For instance, O would be satisfied by a system S_{bad} where the resource proxy releases the resource only in case of failure:

$$S_{bad} = \Gamma \vdash \langle \overline{\mathsf{u}}.\overline{\mathsf{q}} \mid \overline{\mathsf{p}}\natural\overline{\mathsf{v}}.\mathsf{u}.\mathsf{q} \rangle$$

To rule out S_{bad} we can refine O as the observer $O' = \mathsf{p}.(\mathsf{v}.\checkmark + \lightning\overline{\mathsf{u}}.\mathsf{v}.\checkmark)$ that checks if a resource is released (action v) after having been acquired (action p) regardless of possible failures on clients' requests of use (action $\lightning\overline{\mathsf{u}}$). ◇

Notice that observers cannot be composed in parallel and therefore they do not communicate among themselves. This, and the absence of name passing in ATc, allow us to avoid using name restriction in observers. Moreover, observers do not run in transactional scopes and they are not allowed to invoke services. Extending our framework to deal with failures on service invocations would be straightforward. However, we contend that such failures should be dealt with other mechanisms. For instance, a failing invocation may raise an exception which allows the invoker to look for another service provider (rather than triggering a compensation).

Definition 5.2 presents the reduction semantics of observed systems.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Definition 5.2 (semantics of observed systems). The smallest relation $\rightsquigarrow \subseteq \mathsf{Obs} \times \mathsf{Obs}$ satisfying the following axioms and rules yields the semantics of observed systems.

$$\Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\pi \sharp QP \ \S\ R]\!] \parallel \overline{\pi}.O \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![P \ \S\ R \mid Q]\!] \parallel O, \text{ if } \pi, \overline{\pi} \notin \tilde{x} \quad \texttt{(os1)}$$

$$\Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\langle \pi \sharp QP \mid P' \rangle_{Q'} \ \S\ R]\!] \parallel \lightning \pi.O \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![Q' \ \S\ R]\!] \parallel O, \text{ if } \pi, \overline{\pi} \notin \tilde{x} \quad \texttt{(os2)}$$

$$\Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![\pi \sharp QP]\!] \parallel \lightning \pi.O \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![\mathtt{err}]\!] \parallel O, \text{ if } \pi, \overline{\pi} \notin \tilde{x} \quad \texttt{(os3)}$$

$$\frac{S \rightsquigarrow S'}{S \parallel O \rightsquigarrow S' \parallel O} \qquad \frac{S \parallel O \rightsquigarrow S' \parallel O'}{S \parallel O + O'' \rightsquigarrow S' \parallel O'} \quad \texttt{(os4/os5)}$$

$$\frac{S \parallel O_1 \equiv S_1 \parallel O_1 \rightsquigarrow S_2 \parallel O_2 \rightsquigarrow S' \parallel O'}{S \parallel O \rightsquigarrow S' \parallel O'}. \quad \texttt{(os6)}$$

An observer can interact with the system and cause communication failures which trigger the compensations associated with the enclosing transactional scopes (if any). Axiom `(os1)` models a communication step involving (a part of) the system and the observer. Axiom `(os2)` triggers the compensation when there is a communication failure within a transactional scope. Axiom `(os3)` yields an error when a failure occurs outside a transactional scope. Rule `(os4)` models a step due to transitions of the system that does not involve the observer. The interactions of the system with non-deterministic observers are defined by rule `(os5)`. Rule `(os6)` is the usual rule for congruence.

Remark 5.1. In the following, we rule out systems where `err` occurs within transactional scopes. In fact, by our assumption that programmers cannot use `err` and by Definitions 3.5 and 5.2, `err` can be introduced only in scope-avoiding contexts.

5.3. *Typing and observed systems*

Theorem 4.3 can be extended to observed systems. For this we refine the syntax of error processes as follows:

$$\mathtt{err} ::= \mathtt{ierr} \mid \mathtt{cerr}$$

to distinguish between *invocation* (`ierr`) and *communication* (`cerr`) errors. Note that if we replace `err` with `ierr` in rule `(s2)` of Definition 3.6 and `err` with `cerr` in rule `(os3)` of Definition 5.2, then Theorem 4.3 still holds. In the following we then assume that the semantics of systems and observed systems is obtained by replacing `(s2)` and `(os3)` respectively with

$$\Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![s \propto A.P]\!] \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![\mathtt{ierr}]\!], \qquad \text{if } \mathtt{m} \in A \quad \texttt{(s2')}$$

$$\Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![\pi \sharp QP]\!] \parallel \lightning \pi.O \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![\mathtt{cerr}]\!] \parallel O, \text{if } \{\pi, \overline{\pi}\} \cap \tilde{x} = \varnothing. \quad \texttt{(os3')}$$

Moreover, we can prove the following theorem.

Theorem 5.1. If Γ and $P \in \mathcal{P}$ are well-typed and

$$\Gamma \vdash P \parallel O \rightsquigarrow \Gamma \vdash Q \parallel O' \tag{6}$$

then Q does not contain `ierr`.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Proof. The proof is by induction on the derivation of (6). The thesis follows trivially if the proof is an instance of the axioms (os1) or (os2) (as they do not introduce errors). If the proof is an instance of axiom (os3') the only possible error in the right-hand side is a communication error cerr. If the last rule in the proof of (6) is an instance of (os4) the thesis follows by Theorem 4.3. If the last rule in the proof of (6) is an instance of (os5) or (os6) the thesis directly follows by the inductive hypothesis. □

It is worth noticing that the following is a corollary of Theorem 5.1.

Theorem 5.2. If Γ and $P \in \mathcal{P}$ are well-typed and

$$\Gamma \vdash P \parallel O \rightsquigarrow \ldots \rightsquigarrow \Gamma \vdash Q \parallel O'$$

then Q does not contain ierr.

Proof. This result is an immediate consequence of the fact that reductions of observed systems preserve the well-typing. To prove this one can use induction on the structure of derivation or a reduction observing that the only non-trivial case is when the proof ends with an application of rule (os2), say

$$\Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\langle \pi \sharp Q P \mid P' \rangle_{Q'} \fatsemi R]\!] \parallel \not\downarrow_{\pi}.O \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![Q' \fatsemi R]\!] \parallel O$$

(where $\{\pi, \overline{\pi}\} \cap \tilde{x} = \varnothing$). In fact, if the system on the right-hand side is not well-typed, then $\mathbb{C}[\![\natural \fatsemi \boxtimes]\!]$ is scope-avoiding and the type t of Q' should be such that $(\mathfrak{o}, \mathfrak{m}) \notin \widehat{t}$. This yields a contradiction because, by our typing rules (cf. Definition 4.2) the initial system would not be well-typed as its type would contain $\widehat{t}$. □

5.4. *A testing framework for ATc*

Two basic elements of the testing theory in (De Nicola and Hennessy 1984) are the notions of *successful* and *(non-)diverging* computation.

Definition 5.3 (computation). The set Comp (ranged over by c) of *computations* is the set of (possibly infinite) sequences

$$S_0 \parallel O_0, \ldots, S_n \parallel O_n, \ldots \tag{7}$$

such that $S_i \parallel O_i \rightsquigarrow S_{i+1} \parallel O_{i+1}$ for each index i. We say that (7) is a computation starting from $S_0 \parallel O_0$.

Intuitively, a computation is successful if the test is passed (i.e., the corresponding observer halts with ✓). Non-diverging computations are successful computations that reach ✓ before the occurrence of an error.

Definition 5.4 (success and divergence). Let $O \searrow \checkmark$ stand for $O \equiv \checkmark + O'$ for some observer O'. Call *successful* a state $S \parallel O$ where $O \searrow \checkmark$ and call *erroneous* a state $\Gamma \vdash P \parallel O \in \mathsf{Obs}$ when P is erroneous.

— $c = S_0 \parallel O_0, S_1 \parallel O_1, \ldots, S_n \parallel O_n, \ldots \in \mathsf{Comp}$ is *successful* if $S_i \parallel O_i$ is successful for some i (*unsuccessful* otherwise);

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

— c is *diverging* if either c is unsuccessful or there is i such that $S_i \parallel O_i$ is erroneous and $O_j \not\downarrow \checkmark$ for all $j < i$.

The possible outcomes of computations are defined in terms of *result sets* (as customary in testing theory); a result set is a (non-empty) subset of $\{\top, \bot\}$ where $\bot$ and $\top$ denote divergence and non-divergence, respectively.

Definition 5.5 (result set). The *result set of* $S \parallel O \in \mathsf{Obs}$, $\mathfrak{R}(S \parallel O) \subseteq \{\top, \bot\}$, is defined by

— $\top \in \mathfrak{R}(S \parallel O) \iff$ there is a non-diverging $c \in \mathsf{Comp}$ that starts from $S \parallel O$,
— $\bot \in \mathfrak{R}(S \parallel O) \iff$ there is a diverging $c \in \mathsf{Comp}$ that starts from $S \parallel O$.

As in (De Nicola and Hennessy 1984), we consider *may*- and *must*-preorders and the corresponding induced equivalences.

Definition 5.6. 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:

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

The two equivalences $\simeq_{\mathbf{m}}$ and $\simeq_{\mathbf{M}}$ corresponding to $\sqsubseteq_{\mathbf{m}}$ and $\sqsubseteq_{\mathbf{M}}$ are defined as expected

$$\simeq_{\mathbf{m}} \;=\; \sqsubseteq_{\mathbf{m}} \cap \sqsubseteq_{\mathbf{m}}^{-1} \qquad \text{and} \qquad \simeq_{\mathbf{M}} \;=\; \sqsubseteq_{\mathbf{M}} \cap \sqsubseteq_{\mathbf{M}}^{-1}.$$

Example 5.2. Consider the systems S_1 and S_2 in Section 5.1 and the observer O' in Example 5.1. It is immediate from the definitions above that $S_1 \text{ MAY } O'$ and $S_2 \text{ MAY } O'$. Also, $\bot \in \mathfrak{R}(S_2 \parallel O')$, hence $S_2 \text{ MUST } O'$ does not hold, while $S_1 \text{ MUST } O'$ holds true. ◇

6. Comparing configurations of transactional scopes

As shown in previous examples, different attributes enact different behaviours (e.g., Section 5.1); in fact,

i. upon service invocation, different attributes may lead to configurations where the transactional scopes are arranged in a different way, and
ii. different configurations may have different behaviours in case of failure.

While (i) is a direct consequence of the transition rules for systems (Definition 3.6), to illustrate (ii) we give a few examples (cf. Section 6.1) showing that the testing preorders defined for ATc are able to tell apart systems, whose processes have transactional scopes that are differently configured.

We show in Section 6.2 that, under reasonable assumptions on the types of ATc systems, it is possible to relate the behaviour of different configurations of transactional scopes.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

6.1. *Transactional scopes and may-testing*

The following example shows that the behaviour of a system changes depending on how its processes are nested in transactional contexts.

Example 6.1. Consider the following inequalities

$$\Gamma \vdash \langle \overline{x} \mid \overline{y} \rangle_{\overline{z}} \not\sqsubseteq_{\mathbf{m}} \Gamma \vdash \langle \overline{x} \rangle_{\overline{z}} \mid \langle \overline{y} \rangle \quad (8) \qquad \Gamma \vdash \langle \overline{x} \rangle_{\overline{z}} \mid \langle \overline{y} \rangle \not\sqsubseteq_{\mathbf{m}} \Gamma \vdash \langle \overline{x} \mid \overline{y} \rangle_{\overline{z}} \quad (11)$$

$$\Gamma \vdash \langle \overline{x} \| \overline{w} \mid x.\overline{y} \rangle_{\overline{z}} \not\sqsubseteq_{\mathbf{m}} \Gamma \vdash \langle \overline{x} \| \overline{w} \rangle_{\overline{z}} \mid \langle x.\overline{y} \rangle_{\overline{z}} \quad (9) \qquad \Gamma \vdash \langle \overline{x} \rangle_{\overline{z}} \mid \langle \overline{y} \rangle_{\overline{z}} \not\sqsubseteq_{\mathbf{m}} \Gamma \vdash \langle \overline{x} \mid \overline{y} \rangle_{\overline{z}} \quad (12)$$

$$\Gamma \vdash \langle \overline{x} \mid \overline{y} \rangle_{\overline{z}} \not\sqsubseteq_{\mathbf{m}} \Gamma \vdash \langle \overline{x} \rangle_{\overline{z}} \mid \overline{y} \quad (10) \qquad \Gamma \vdash \langle \overline{x} \rangle_{\overline{z}} \mid \overline{y} \not\sqsubseteq_{\mathbf{m}} \Gamma \vdash \langle \overline{x} \mid \overline{y} \rangle_{\overline{z}}. \quad (13)$$

It is straightforward to verify that

— $\lightning y.z.\checkmark$ distinguishes the systems in (8) and (10);
— $\lightning y.w.\checkmark$ distinguishes the systems in (9);
— $\lightning x.y.\checkmark$ distinguishes the systems in (11), (12) and (13).

In fact, the observers are successful for the systems on the left-hand side of the inequalities, and fail on the others. ◇

Example 6.1 yields just some instances of the fact that, in general,

$$\Gamma \vdash \langle P \mid R \rangle_Q \not\sqsubseteq_{\mathbf{m}} \Gamma \vdash \langle P \rangle_Q \mid \langle R \rangle \qquad \Gamma \vdash \langle P \mid R \rangle_Q \not\sqsupseteq_{\mathbf{m}} \Gamma \vdash \langle P \rangle_Q \mid \langle R \rangle \quad (14)$$

$$\Gamma \vdash \langle P \mid R \rangle_Q \not\sqsubseteq_{\mathbf{m}} \Gamma \vdash \langle P \rangle_Q \mid \langle R \rangle_Q \qquad \Gamma \vdash \langle P \mid R \rangle_Q \not\sqsupseteq_{\mathbf{m}} \Gamma \vdash \langle P \rangle_Q \mid \langle R \rangle_Q \quad (15)$$

$$\Gamma \vdash \langle P \mid R \rangle_Q \not\sqsubseteq_{\mathbf{m}} \Gamma \vdash \langle P \rangle_Q \mid R \qquad \Gamma \vdash \langle P \mid R \rangle_Q \not\sqsupseteq_{\mathbf{m}} \Gamma \vdash \langle P \rangle_Q \mid R. \quad (16)$$

Inequalities (14) and (15) highlight that transactional scopes do not distribute with parallel composition. Moreover, by (16), transactional scopes do not commute with the parallel operator and compensations do not distribute over scopes. Notice that similar inequalities hold for $\sqsubseteq_{\mathbf{M}}$ as well.

6.2. *Prudent systems and structure preservation*

Intuitively, $\Gamma \vdash \langle P \rangle_Q \mid R \parallel O$ behaves similarly to $\Gamma \vdash P \mid R \parallel O$ until a failure occurs or until a service invocation causes different scope reconfigurations in the two systems.

Proposition 6.1. Given $\Gamma \vdash \langle P \rangle_Q \mid R \parallel O \in \mathsf{Obs}$, if there is $S \parallel O' \in \mathsf{Obs}$ non-diverging,

$$\Gamma \vdash \langle P \rangle_Q \mid R \parallel O \quad \rightsquigarrow \quad S \parallel O' \qquad (17)$$

then $S = \Gamma \vdash \langle P' \rangle_Q \mid R' \parallel O'$ or $S = \Gamma \vdash Q \mid R \parallel O'$.

Proof. The proof is by induction on the derivation of (17) considering that axiom (os3) cannot be applied otherwise $S \parallel O'$ would be diverging (see appendix B.1). □

Later in this section, we show that if attributes are used 'wisely', then it is possible to use may-testing to compare the behaviours of two systems where the same process executes in different transactional contexts. Such results are based on the notions of *structure preserving attributes* and *prudent systems*, namely systems in which the configuration of the transactional scopes is preserved through reduction.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Table 2. *Evolution of transactional scopes wrt attributes, with* $\hat{P} \in \Gamma(s, a)$.

Preserving attributes		
$a = \mathtt{s}$	$a = \mathtt{rn}$	$a = \mathtt{ns}$
$\Gamma \vdash \mathbf{A}[\![P]\!] \rightsquigarrow \Gamma \vdash \mathbf{A}[\![P'' \mid \hat{P}]\!]$	$\Gamma \vdash \mathbf{A}[\![P]\!] \rightsquigarrow \Gamma \vdash \mathbf{A}[\![P'']\!] \mid \langle \hat{P} \rangle$	$\Gamma \vdash \mathbf{A}[\![P]\!] \rightsquigarrow \Gamma \vdash \mathbf{A}[\![P'']\!] \mid \hat{P}$
$\Gamma \vdash \mathbf{A}[\![\langle P \rangle_Q]\!] \rightsquigarrow \Gamma \vdash \mathbf{A}[\![\langle P'' \mid \hat{P} \rangle_Q]\!]$	$\Gamma \vdash \mathbf{A}[\![\langle P \rangle_Q]\!] \rightsquigarrow \Gamma \vdash \mathbf{A}[\![\langle P'' \rangle_Q]\!] \mid \langle \hat{P} \rangle$	$\Gamma \vdash \mathbf{A}[\![\langle P \rangle_Q]\!] \rightsquigarrow \Gamma \vdash \mathbf{A}[\![\langle P'' \rangle_Q]\!] \mid \hat{P}$
Non-preserving attributes		
$a = \mathtt{n}$	$a = \mathtt{r}$	$a = \mathtt{m}$
$\Gamma \vdash \mathbf{A}[\![P]\!] \rightsquigarrow \Gamma \vdash \mathbf{A}[\![P'']\!] \mid \hat{P}$	$\Gamma \vdash \mathbf{A}[\![P]\!] \rightsquigarrow \Gamma \vdash \mathbf{A}[\![P'']\!] \mid \langle \hat{P} \rangle$	$\Gamma \vdash \mathbf{A}[\![P]\!] \rightsquigarrow \Gamma \vdash \mathbf{A}[\![\mathtt{err}]\!]$
$\Gamma \vdash \mathbf{A}[\![\langle P \rangle_Q]\!] \rightsquigarrow \Gamma \vdash \mathbf{A}[\![Q]\!]$	$\Gamma \vdash \mathbf{A}[\![\langle P \rangle_Q]\!] \rightsquigarrow \Gamma \vdash \mathbf{A}[\![\langle P'' \mid \hat{P} \rangle_Q]\!]$	$\Gamma \vdash \mathbf{A}[\![\langle P \rangle_Q]\!] \rightsquigarrow \Gamma \vdash \mathbf{A}[\![\langle P'' \mid \hat{P} \rangle_Q]\!]$

Consider the systems

$$S_1 = \Gamma \vdash P \mid R \qquad \text{and} \qquad S_2 = \Gamma \vdash \langle P \rangle_Q \mid R \tag{18}$$

where $P = s \propto a.P''$ invokes a service s with an (unspecified) attribute a and then behaves like P''. The invocation of s launches the execution of $\hat{P}$ (i.e., $\hat{P} \in \Gamma(s, a)$). We are interested in discerning for which transactional attributes (i.e., values of a) S_1 and S_2 *preserve their structure*, namely when, for some P', $\mathbf{A'}[\![\square]\!]$:

$$\Gamma \vdash \nu \tilde{x} \mathbf{A}[\![P]\!] \rightsquigarrow \Gamma \vdash \nu \tilde{x} \mathbf{A'}[\![P']\!] \quad \Longrightarrow \quad \Gamma \vdash \nu \tilde{x} \mathbf{A}[\![\langle P \rangle_Q]\!] \rightsquigarrow \Gamma \vdash \nu \tilde{x} \mathbf{A'}[\![\langle P' \rangle_Q]\!].$$

Table 2 summarises what happens depending on the value of a. If $a \in \{\mathtt{s}, \mathtt{ns}, \mathtt{rn}\}$ then the structure of both S_1 and S_2 is preserved:

— if $a = \mathtt{s}$, the service instance 'adapts' to the structure of the invoker's scope;
— if $a = \mathtt{ns}$ then the service instance behaves as in the previous case for non-transactional invocations, otherwise it can be 'assembled' with R preserving the same structure of (18) (i.e., $\mathbf{A'}[\![\square]\!] = \mathbf{A}[\![\square]\!] \mid \langle \hat{P} \rangle$);
— the case $a = \mathtt{rn}$ is similar to the case $a = \mathtt{ns}$ but for the fact that the service instance is executed in a new transactional scope (i.e., $\mathbf{A'}[\![\square]\!] = \mathbf{A}[\![\square]\!] \mid \hat{P}$).

Instead, if $a \in \{\mathtt{n}, \mathtt{r}, \mathtt{m}\}$ then the structure of either of S_1 and S_2 is not preserved:

— if $a = \mathtt{n}$ only $\Gamma \vdash P \mid R$ moves to a configuration that includes the service instance $\hat{P}$ whereas $\Gamma \vdash \langle P \rangle_Q \mid R$ triggers the compensation process;
— if $a = \mathtt{r}$ then the transactional scope of $\hat{P}$ in the transactional invocation is different from the existing one (i.e., there are no $\mathbf{A'}[\![\square]\!]$ and P' for which the configurations of P and $\langle P \rangle_Q$ are preserved after the invocation);
— if $a = \mathtt{m}$ only in the transactional invocation the configuration includes a new instance of the invoked service $\hat{P}$ whereas, in the other case, it includes an erroneous process.

An interesting class of systems can be characterised in terms of the types for ATc defined in Section 4.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Definition 6.1 (prudent systems). A well-typed process $P \in \mathcal{P}$ is *prudent* iff $P \triangleright t$ and $(\mathtt{o},\mathtt{n}),(\mathtt{o},\mathtt{r}) \notin \widehat{t}$. A system $\Gamma \vdash P$ is *prudent* if P is prudent and, for each Q such that $(s,a,Q) \in \Gamma$ for some $s \in \mathcal{S}$ and $a \in \mathcal{A}$, Q is prudent.

By definition, processes in prudent systems can use only 'structure preserving' attributes (notice that $(\mathtt{o},\mathtt{m})$ is also ruled out by well-typedness of P). Consider the systems S_1 and S_2 in (18); the assumption that they are prudent allows us to say that each transition to non-erroneous processes of S_1 corresponds to a transition of S_2 that preserves the relationship between the configurations. More formally, a prudent process preserves its behaviour in any parallel context when it is executed in a transactional scope with any compensation.

Lemma 6.1. Let $\Gamma \vdash P$ be a prudent system. For any observer O and any $R \in \mathcal{P}$

$$\Gamma \vdash P \mid R \parallel O \rightsquigarrow \Gamma \vdash P' \mid R' \parallel O' \quad \Longrightarrow \quad \Gamma \vdash \langle P \rangle_Q \mid R \parallel O \rightsquigarrow S \parallel O' \tag{19}$$

where $S = \Gamma \vdash \langle P' \rangle_Q \mid R'$ or $S = \Gamma \vdash Q \mid R'$.

Proof. The proof is by induction on the derivation of the transition of the hypothesis in the implication (19). The full proof is relegated in appendix B.2. □

Theorem 6.1. If $\Gamma \vdash P$ is a prudent system then

$$\Gamma \vdash P \mid R \quad \sqsubseteq_{\mathbf{m}} \quad \Gamma \vdash \langle P \rangle_Q \mid R$$

for all $Q, R \in \mathcal{P}$.

Proof. Let O_0 be an observer, $P_0 = P \mid R$, and

$$\Gamma \vdash P_0 \parallel O_0, \ldots, \Gamma \vdash P_i \parallel O_i \tag{20}$$

be a non-diverging computation of $\Gamma \vdash P$ such that $\Gamma \vdash P_i \parallel O_i$ is a successful state.

Since all the states in (20) are non-erroneous (because (20) is non-diverging), by induction on i, we exhibit a non-diverging from $\Gamma \vdash \langle P \rangle_Q \mid R \parallel O_0$ with the same observers, using Lemma 6.1. □

7. Testing-preserving attribute substitutions

In this section we apply the testing framework of ATc to prove that, under suitable conditions, attributes are interchangeable. Namely, using an attribute instead of another in a service invocation does not alter the observational behaviour of systems.

Example 7.1. The example in Section 5.1 illustrates how different attributes yield different observable behaviours, however, the observational behaviour of $Q = \langle s \propto \mathtt{r}.\overline{\mathtt{u}}.\overline{\mathtt{q}} \rangle$ does not change by replacing $\mathtt{r}$ with $\mathtt{m}$ (which yields the process P_1 in Section 5.1) or $\mathtt{s}$. ◇

We investigate a preorder relation on attributes defined in terms of the may-testing preorder and show that systems exhibit the same behaviour when all occurrences of an attribute are replaced with an equivalent one. More precisely, we define the preorder $\leqslant_{\mathbf{m}} \subseteq \mathcal{A} \times \mathcal{A}$, (**m** stands for *may*) that correspond to the preorder $\sqsubseteq_{\mathbf{m}}$ for systems. The intuition is that if S' is obtained by replacing b for a in a system S and $a \leqslant_{\mathbf{m}} b$, then $S \sqsubseteq_{\mathbf{m}} S'$. Preorder $\leqslant_{\mathbf{m}}$ is defined in terms of preorders $\leqslant^{\mathtt{o}}_{\mathbf{m}}$ and $\leqslant^{\mathtt{i}}_{\mathbf{m}}$ which differ on

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

whether the replacement of the attributes affects only the calls done outside or inside a transactional scope, respectively.

Before introducing $\leqslant_{\mathbf{m}}$ (cf. Definition 7.3), we give some auxiliary definitions and results on substitutions of transactional attributes.

7.1. *Attribute substitutions*

We consider two special notions of substitutions of attributes; one replaces attributes that appear in transactional invocations and the other replaces those outside transactional scopes. Let $[^b/_a]$ denote a *standard substitution*, namely a substitution that replaces any occurrence of a with b; given $A \subseteq \mathcal{A}$, let $[^b/_a]A$ abbreviate $\{[^b/_a](a') | a' \in A\}$.

Definition 7.1 (attribute substitution on $\mathcal{P}$). The action of substitutions $[^b/_a]^{\mathrm{o}}$ and $[^b/_a]^{\mathrm{i}}$ on processes are defined as follows:

$$
\begin{array}{llll}
\varnothing[^b/_a]^{\mathrm{o}} & = \varnothing & \varnothing[^b/_a]^{\mathrm{i}} & = \varnothing \\
(\nu x\ P)[^b/_a]^{\mathrm{o}} & = \nu x\ (P[^b/_a]^{\mathrm{o}}) & (\nu x\ P)[^b/_a]^{\mathrm{i}} & = \nu x\ (P[^b/_a]^{\mathrm{i}}) \\
(P \mid Q)[^b/_a]^{\mathrm{o}} & = P[^b/_a]^{\mathrm{o}} \mid Q[^b/_a]^{\mathrm{o}} & (P \mid Q)[^b/_a]^{\mathrm{i}} & = P[^b/_a]^{\mathrm{i}} \mid Q[^b/_a]^{\mathrm{i}} \\
(!P)[^b/_a]^{\mathrm{o}} & = !P[^b/_a]^{\mathrm{o}} & (!P)[^b/_a]^{\mathrm{i}} & = !P[^b/_a]^{\mathrm{i}} \\
(\pi \Vdash Q.P)[^b/_a]^{\mathrm{o}} & = \pi \Vdash Q[^b/_a]^{\mathrm{o}}.P[^b/_a]^{\mathrm{o}} & (\pi \Vdash Q.P)[^b/_a]^{\mathrm{i}} & = \pi \Vdash Q[^b/_a]^{\mathrm{i}}.P[^b/_a]^{\mathrm{i}} \\
(s \propto A.P)[^b/_a]^{\mathrm{o}} & = s \propto [^b/_a]A.(P[^b/_a]^{\mathrm{o}}) & (s \propto A.P)[^b/_a]^{\mathrm{i}} & = s \propto A.(P[^b/_a]^{\mathrm{i}}) \\
\langle P \rangle_Q [^b/_a]^{\mathrm{o}} & = \langle P \rangle_{Q[^b/_a]^{\mathrm{o}}} & \langle P \rangle_Q [^b/_a]^{\mathrm{i}} & = \langle P[^b/_a] \rangle_{Q[^b/_a]^{\mathrm{i}}}.
\end{array}
$$

Hereafter, we denote any substitution (standard or not) with σ and define $\mathtt{err}\sigma = \mathtt{err}$.

Intuitively, $[\text{-}/_]^{\mathrm{o}}$ acts as the standard substitution except for the case of transactional scope where 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 $[\text{-}/_]^{\mathrm{i}}$ acts as expected but for service invocation and transactional scopes. In the former case, the substitution does not change the attributes of the invocation to s, since the process is outside a transactional scope; in the latter case, the standard substitution of a for b is applied to the running process of the scope P (the substitutions have 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).

Definition 7.2 (attribute substitution on systems). The action of an attribute substitution σ on a system $\Gamma \vdash P$ is defined as $(\Gamma \vdash P)\sigma = \Gamma\sigma \vdash P\sigma$ where

$$\Gamma\sigma = \bigcup_{(s,a,P) \in \Gamma} \{(s, a, P), (s, \sigma(a), P\sigma), (s, a, P\sigma)\}.$$

Moreover, $(S \parallel O)\sigma = S\sigma \parallel O$ for any attribute substitution σ.

The substitution on Γ replicates a service description so that the body is available for both invocations of s done with a and with $\sigma(a)$. For example, if P makes a transactional invocation to s with attribute a, then $P[^b/_a]^{\mathrm{i}}$ must be able to make a corresponding transactional invocation of s with attribute b. Furthermore, the environment must still be able to offer a service for attribute a as $P[^b/_a]^{\mathrm{i}}$ may still include non-transactional invocations with attribute a.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Proposition 7.1. Attribute substitutions are idempotent.

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

Lemma 7.1. For any system S and any $a \in \mathcal{A}$, $S[^{a}/_{a}]^{\text{o}} = S[^{a}/_{a}]^{\text{i}} = S[^{a}/_{a}] = S$.

Proof. By induction on the structure of S. □

7.2. *Attribute substitutions and reduction semantics*

We consider the relationship between attribute substitutions on the one hand and the reduction semantics of processes, systems, and observed systems on the other hand. The results of this section will be used in Section 7.3 to prove the equivalence and order relations on transactional attributes.

We first show that attribute substitutions preserve processes' reductions.

Proposition 7.2. Given an ATc process P

$$P \to Q \implies P\sigma \to Q\sigma$$

holds for any attribute substitution σ.

Proof. The proof easily follows by observing that the reduction semantics of processes (cf. Definition 3.4, 627) does not depend on attributes. □

For systems the situation is different; the reduction semantics of ATc systems does depend on the transactional attributes used in service invocations or those supported in containers. We focus on substitutions that will be used in the proof of Theorem 7.2.

Proposition 7.3. Let S be a prudent system such that $S \rightsquigarrow S'$ for a system S'. If $\sigma \in \{[^{\mathtt{rn}}/_{\mathtt{ns}}]^{\text{o}}\} \cup \{[^{\mathtt{b}}/_{\mathtt{a}}]^{\text{o}} \mid a \in \{\mathtt{m},\mathtt{n},\mathtt{r}\} \wedge b \in \mathcal{A}\} \cup \{[^{\mathtt{b}}/_{\mathtt{a}}]^{\text{o}} \mid a,b \in \{\mathtt{n},\mathtt{s},\mathtt{ns}\}\}$ then there is S'' such that:

$$S\sigma \rightsquigarrow S'' \quad \text{and} \quad S'\sigma \sqsubseteq_{\mathbf{m}} S''. \tag{21}$$

Proof. By case analysis on the derivation of $S \rightsquigarrow S'$ as per Definition 3.6 (page 629). The proof is relegated in appendix C.1. □

In some cases, Proposition 7.3 can be generalised as attribute substitutions preserve transitions of systems as shown by Corollary 7.1.

Corollary 7.1. If $a,b \in \{\mathtt{n},\mathtt{ns},\mathtt{s}\}$,

$$S \rightsquigarrow S' \qquad \implies \qquad S[^{b}/_{a}]^{\text{o}} \rightsquigarrow S'[^{b}/_{a}]^{\text{o}}$$

for any system S.

Proof. By inspection of the proof of Proposition 7.3 when $\sigma = [^{b}/_{a}]^{\text{o}}$ with $a,b \in \{\mathtt{n},\mathtt{ns},\mathtt{s}\}$, observing that $S\sigma \rightsquigarrow S'\sigma$ and the hypothesis that S is prudent is not necessary in none of the cases. □

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Furthermore, the following corollary shows that we obtain a similar property by allowing non-transactional invocations with attribute $\mathtt{m}$.

Corollary 7.2. Let $\sigma = [^{a}/_{\mathtt{m}}]^{\mathtt{o}}$ with $a \in \mathcal{A}$. For any system S, if $S \rightsquigarrow S'$ then there exists S'' such that

$$(S\sigma \rightsquigarrow S'' \wedge S'\sigma \sqsubseteq_{\mathbf{m}} S'') \quad \vee \quad S'\sigma \sqsubseteq_{\mathbf{m}} S\sigma.$$

Proof. By inspection of the proof of Proposition 7.3 when $\sigma = [^{a}/_{\mathtt{m}}]^{\mathtt{o}}$, observing that the case of non-transactional invocations with attribute $\mathtt{m}$ (which in the proof of Proposition 7.3 is ruled out) satisfies $S'\sigma \sqsubseteq_{\mathbf{m}} S\sigma$ (i.e., after the reduction with rule (s2) S would move to an erroneous state). □

Proposition 7.4. Let σ be either of the substitutions $[^{\mathtt{rn}}/_{\mathtt{ns}}]^{\mathtt{i}}$ or $[^{b}/_{a}]^{\mathtt{i}}$ with $a, b \in \{\mathtt{r}, \mathtt{s}, \mathtt{m}\}$, then

$$S \rightsquigarrow S' \implies S\sigma \rightsquigarrow S'' \text{ with } S'\sigma \sqsubseteq_{\mathbf{m}} S''$$

for any system S.

Proof. Similarly to the proof of Proposition 7.3, the proof proceeds by cases on the derivation of $S \rightsquigarrow S'$, and is relegated in appendix C.2. □

Corollary 7.3 yields an analogous result as Corollary 7.1.

Corollary 7.3. If $a, b \in \{\mathtt{r}, \mathtt{s}, \mathtt{m}\}$

$$S \rightsquigarrow S' \qquad \Longrightarrow \qquad S[^{b}/_{a}]^{\mathtt{i}} \quad \rightsquigarrow \quad S'[^{b}/_{a}]^{\mathtt{i}}$$

for any system S.

Proof. By inspection of the proof of Proposition 7.4 when $\sigma = [^{b}/_{a}]^{\mathtt{i}}$ with $a, b \in \{\mathtt{r}, \mathtt{s}, \mathtt{m}\}$, observing that $S\sigma \rightsquigarrow S'\sigma$. □

Lemma 7.2. Let $\mathfrak{t}$ be a transition $S \parallel O \rightsquigarrow S' \parallel O'$ of an observed system $S \parallel O$. Then

$$(S \parallel O)[^{b}/_{a}]^{\mathtt{o}} \rightsquigarrow (S' \parallel O')[^{b}/_{a}]^{\mathtt{o}} \qquad \text{if } a, b \in \{\mathtt{n}, \mathtt{ns}, \mathtt{s}\}$$
$$(S \parallel O)[^{b}/_{a}]^{\mathtt{i}} \rightsquigarrow (S' \parallel O')[^{b}/_{a}]^{\mathtt{i}} \qquad \text{if } a, b \in \{\mathtt{r}, \mathtt{s}, \mathtt{m}\}.$$

Proof. The proof is by induction on the derivation of $\mathfrak{t}$ (see appendix C.3). □

7.3. *Ordering transactional attributes*

Attribute substitutions and may-preorder on systems induce preorders on transactional attributes as per Definition 7.3.

Definition 7.3 (attribute order). Let $a, b \in \mathcal{A}$ and S be a system. We define

$$a \leqslant^{\mathtt{o}}_{\mathbf{m}} b \iff S \sqsubseteq_{\mathbf{m}} S[^{b}/_{a}]^{\mathtt{o}} \qquad \text{and} \qquad a \leqslant^{\mathtt{i}}_{\mathbf{m}} b \iff S \sqsubseteq_{\mathbf{m}} S[^{b}/_{a}]^{\mathtt{i}}$$

also, $\leqslant_{\mathbf{m}} \stackrel{def}{=} \leqslant^{\mathtt{i}}_{\mathbf{m}} \cap \leqslant^{\mathtt{o}}_{\mathbf{m}}$ and $=_{\mathbf{m}} \stackrel{def}{=} \leqslant_{\mathbf{m}} \cap \leqslant^{-1}_{\mathbf{m}}$ (similar definitions are assumed for $=^{\mathtt{o}}_{\mathbf{m}}$ and $=^{\mathtt{i}}_{\mathbf{m}}$).

Figure 2 illustrates the relationships among attributes for the order relations in Definition 7.3; note that

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

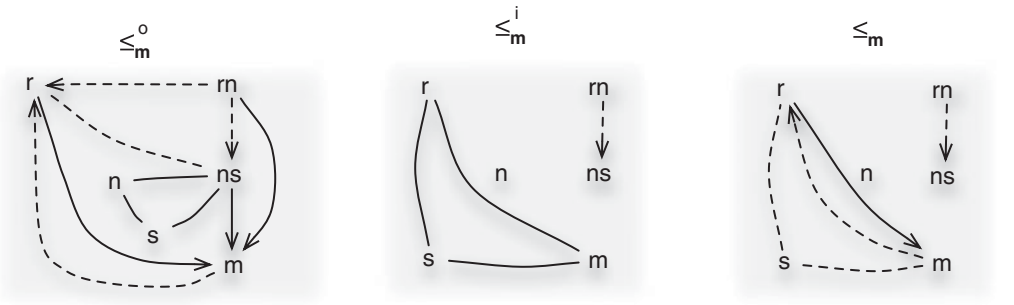


Fig. 2. Order on attributes (the dashed arrows assume prudent systems).

— dashed edges hold only for prudent systems (cf. Theorem 7.2) and point at the smaller element (e.g., in the leftmost box, $\mathtt{r} \leftarrow \mathtt{rn}$ reads as $\mathtt{r} \leqslant^{o}_{m} \mathtt{rn}$);
— unarrowed edges represent the equality corresponding to the preorder (e.g., in the leftmost box, $\mathtt{s}$—$\mathtt{n}$ stands for $\mathtt{s} =^{o}_{m} \mathtt{n}$) and correspond to the equivalences in Theorem 7.1.

Theorems 7.1 and 7.2 below give a formal account of Figure 2.

Theorem 7.1. The following equivalences on $\mathcal{A}$ hold

$$\mathtt{ns} =^{o}_{m} \mathtt{n} \quad =^{o}_{m} \quad \mathtt{s} \tag{22}$$

$$\mathtt{m} =^{i}_{m} \mathtt{r} \quad =^{i}_{m} \quad \mathtt{s}. \tag{23}$$

Proof. Assume that there is a successful computation c from $S \parallel O$. By induction on the length of c and Lemma 7.2 it follows that there is a successful computation from $(S \parallel O)\sigma$ for any $\sigma \in \{[^{b}/_{a}]^{o} | a, b \in \{\mathtt{n}, \mathtt{ns}, \mathtt{s}\}\} \cup \{[^{b}/_{a}]^{i} | a, b \in \{\mathtt{r}, \mathtt{s}, \mathtt{m}\}\}$. □

Theorem 7.2. Consider the order relations

(a) $\mathtt{ns} \leqslant^{o}_{m} \mathtt{rn}$ (b) $\mathtt{m} \leqslant^{o}_{m} a, \quad a \in \mathcal{A}$
(c) $a \leqslant^{o}_{m} b, \quad a \in \{\mathtt{n}, \mathtt{r}\}, b \in \mathcal{A}$ (d) $\mathtt{ns} \leqslant^{i}_{m} \mathtt{rn}$.

We have that (a), (c) and (d) hold for prudent systems, while (b) holds for any system.

Proof. By induction on the derivation of $S \parallel O \rightsquigarrow S' \parallel O'$ (see appendix C.4). □

7.4. *The missing arrows*

In this section we discuss why some arrows are missing from Figure 2. Below we write $a \neq^{i}_{m} b$ to denote $a \not\leqslant^{i}_{m} b$ or $a \not\geqslant^{i}_{m} b$ (and similarly for $a \neq^{o}_{m} b$).

Figure 3 explicitly illustrates the arrows that are missing from Figure 2, representing them as dotted arrows. The numbering on the arrows refers to the counterexamples provided in the rest of this section for both $[^{-}/_{_}]^{o}$ and $[^{-}/_{_}]^{i}$ substitutions. The attributes $\mathtt{ns}$, $\mathtt{n}$, and $\mathtt{s}$ have been grouped together in the left-hand side diagram as they are equivalent with respect to $=^{o}_{m}$. The same conventions are adopted for $\mathtt{r}$, $\mathtt{s}$, and $\mathtt{m}$ and $=^{i}_{m}$ in the right-hand side diagram.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

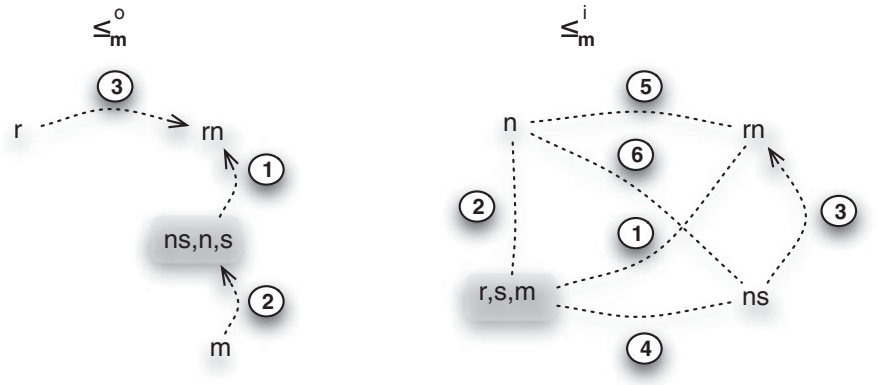


Fig. 3. Inequalities that do not hold in Figure 2.

We first consider the missing arrows for $[^{-}/_{_}]^{o}$ substitutions, that is

1. $\mathtt{rn} \not\leq^{o}_{\mathtt{m}} \mathtt{ns}$, $\mathtt{rn} \not\leq^{o}_{\mathtt{m}} \mathtt{n}$, and $\mathtt{rn} \not\leq^{o}_{\mathtt{m}} \mathtt{s}$
2. $\mathtt{ns} \not\leq^{o}_{\mathtt{m}} \mathtt{m}$, $\mathtt{n} \not\leq^{o}_{\mathtt{m}} \mathtt{m}$, and $\mathtt{s} \not\leq^{o}_{\mathtt{m}} \mathtt{m}$
3. $\mathtt{rn} \not\leq^{o}_{\mathtt{m}} \mathtt{r}$.

For each of the inequalities above, we consider the corresponding attribute substitution σ and exhibit a system S and observer O such that S passes the test while $S\sigma$ does not.

(7.4) Let $\sigma = [^{\mathtt{ns}}/_{\mathtt{rn}}]^{o}$, $\Gamma = \{(s, \mathtt{rn}, y), (s, \mathtt{ns}, y)\}$, and $S = \Gamma \vdash s \propto \mathtt{rn}.x$. Then

$$\Gamma\sigma = \Gamma \quad \text{and} \quad S\sigma = \Gamma \vdash s \propto \mathtt{ns}.x$$

After invoking s, S and $S\sigma$ reduce respectively to $\Gamma \vdash x \mid \langle y \rangle$ and $\Gamma \vdash x \mid y$. Hence, S passes the test $\lightning y.\overline{x}.\checkmark$ while $S\sigma$ fails it.

The inequalities $\mathtt{rn} \not\leq^{o}_{\mathtt{m}} \mathtt{n}$ and $\mathtt{rn} \not\leq^{o}_{\mathtt{m}} \mathtt{s}$ follow by contradiction from $\mathtt{ns} =^{o}_{\mathtt{m}} \mathtt{n} =^{o}_{\mathtt{m}} \mathtt{s}$ (cf. (22) in Theorem 7.1) and $\mathtt{rn} \not\leq^{o}_{\mathtt{m}} \mathtt{ns}$.

(7.4) Let $\sigma = [^{\mathtt{m}}/_{\mathtt{ns}}]^{o}$, $\Gamma = \{(s, \mathtt{m}, y), (s, \mathtt{ns}, y)\}$, and $S = \Gamma \vdash s \propto \mathtt{ns}.x$. Then

$$\Gamma = \Gamma\sigma \quad \text{and} \quad S\sigma = \Gamma \vdash s \propto \mathtt{m}.x$$

After invoking s, S and $S\sigma$ reduce respectively to $\Gamma \vdash x \mid y$ and $\Gamma \vdash \mathtt{err}$. Hence, S passes the test $\overline{x}.\checkmark$ while $S\sigma$ fails it.

The inequalities $\mathtt{n} \not\leq^{o}_{\mathtt{m}} \mathtt{m}$ and $\mathtt{s} \not\leq^{o}_{\mathtt{m}} \mathtt{m}$ follow by contradiction from $\mathtt{ns} =^{o}_{\mathtt{m}} \mathtt{n} =^{o}_{\mathtt{m}} \mathtt{s}$ (cf. (22) in Theorem 7.1) and $\mathtt{ns} \not\leq^{o}_{\mathtt{m}} \mathtt{m}$.

(7.4) Let $\sigma = [^{\mathtt{r}}/_{\mathtt{rn}}]^{o}$, $\Gamma = \{(s, \mathtt{rn}, s' \propto \mathtt{rn}.x), (s', \mathtt{rn}, y)\}$, and $S = \Gamma \vdash s \propto \mathtt{rn}$. Then

$$\Gamma\sigma = \Gamma \cup \{(s, \mathtt{r}, s \propto \mathtt{r}.x), (s, \mathtt{rn}, s \propto \mathtt{r}.x), (s', \mathtt{r}, y)\} \quad \text{and} \quad S\sigma = \Gamma\sigma \vdash s \propto \mathtt{r}.$$

After invoking s, S and $S\sigma$ reduce respectively as follows:

$$\begin{aligned} S &= \Gamma \vdash s \propto \mathtt{rn} \rightsquigarrow \Gamma \vdash \langle s' \propto \mathtt{rn}.x \rangle \rightsquigarrow \Gamma \vdash \langle x \rangle \mid \langle y \rangle \\ S\sigma &= \Gamma\sigma \vdash s \propto \mathtt{r} \rightsquigarrow \Gamma\sigma \vdash \langle s' \propto \mathtt{r}.x \rangle \rightsquigarrow \Gamma\sigma \vdash \langle x \mid y \rangle. \end{aligned}$$

Hence, S passes the test $\lightning x.\overline{y}.\checkmark$ while $S\sigma$ fails it.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

We now turn our attention to the missing arrows for $[\cdot/_]^{\mathrm{i}}$ substitutions, namely

1. $\mathtt{r} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{rn}$, $\mathtt{s} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{rn}$, and $\mathtt{m} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{rn}$
2. $\mathtt{r} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{n}$, $\mathtt{s} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{n}$, and $\mathtt{m} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{n}$
3. $\mathtt{rn} \not\leqslant^{\mathrm{i}}_{\mathbf{m}} \mathtt{ns}$
4. $\mathtt{r} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{ns}$, $\mathtt{s} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{ns}$, and $\mathtt{m} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{ns}$
5. $\mathtt{rn} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{n}$
6. $\mathtt{ns} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{n}$.

For each of the inequalities above but (7.4), we consider the corresponding attribute substitution σ together with its inverse and exhibit two systems S_1 and S_2 such that $S_1 = S_2\sigma$ and $S_2 = S_1\sigma^{-1}$ together with two observers such that each S_i passes one test but not the other. For (7.4), we proceed as for the missing arrows for $[\cdot/_]^{\mathrm{o}}$ substitutions.

(7.4) Assume $\Gamma = \{(s, \mathtt{r}, y), (s, \mathtt{rn}, y)\}$ and consider

$$S_1 = \Gamma \vdash \langle s \propto \mathtt{r}.x \rangle_z \quad \text{and} \quad S_2 = \Gamma \vdash \langle s \propto \mathtt{rn}.x \rangle_z$$

(note that $S_1 = S_2[^{\mathtt{rn}}/_{\mathtt{r}}]^{\mathrm{i}}$ and $S_2 = S_1[^{\mathtt{r}}/_{\mathtt{rn}}]^{\mathrm{i}}$). After invoking s, S_1 and S_2 respectively reduce to $\Gamma \vdash \langle x \mid y \rangle_z$ and $\Gamma \vdash \langle x \rangle_z \mid \langle y \rangle$. Hence, only S_1 passes the test $\lightning y.\overline{z}.\checkmark$ while only S_2 passes the test $\lightning x.\overline{y}.\checkmark$.

The inequalities $\mathtt{s} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{rn}$ and $\mathtt{m} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{rn}$ follow by contradiction from $\mathtt{r} =^{\mathrm{i}}_{\mathbf{m}} \mathtt{s} =^{\mathrm{i}}_{\mathbf{m}} \mathtt{m}$ (cf. (23) in Theorem 7.1) and $\mathtt{r} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{rn}$.

(7.4) Assume $\Gamma = \{(s, \mathtt{r}, \overline{x}), (s, \mathtt{n}, \overline{x})\}$ and consider

$$S_1 = \Gamma \vdash \langle s \propto \mathtt{r} \rangle_z \quad \text{and} \quad S_2 = \Gamma \vdash \langle s \propto \mathtt{n} \rangle_z$$

(note that $S_1 = S_2[^{\mathtt{r}}/_{\mathtt{n}}]^{\mathrm{i}}$ and $S_2 = S_1[^{\mathtt{n}}/_{\mathtt{r}}]^{\mathrm{i}}$). After invoking s, S_1 and S_2 respectively reduce $\Gamma \vdash \langle \overline{x} \rangle_z$ and $\Gamma \vdash z$. Hence, only S_1 passes the test $\lightning \overline{x}.\overline{z}.\checkmark$, while only S_2 passes $\overline{z}.\checkmark$.

The inequalities $\mathtt{s} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{n}$ and $\mathtt{m} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{n}$ follow by contradiction from $\mathtt{r} =^{\mathrm{i}}_{\mathbf{m}} \mathtt{s} =^{\mathrm{i}}_{\mathbf{m}} \mathtt{m}$ (cf. (23) in Theorem 7.1) and $\mathtt{r} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{n}$.

(7.4) Assume $\Gamma = \{(s, \mathtt{rn}, x), (s, \mathtt{ns}, x)\}$ and consider

$$S_1 = \Gamma \vdash \langle s \propto \mathtt{rn} \rangle_{\overline{z}} \quad \text{and} \quad S_2 = \Gamma \vdash \langle s \propto \mathtt{ns} \rangle_{\overline{z}}$$

(note that $S_2 = S_1[^{\mathtt{ns}}/_{\mathtt{rn}}]^{\mathrm{i}}$). After invoking s, S_1 and S_2 respectively reduce to $\Gamma \vdash \langle x \rangle$ and $\Gamma \vdash x$. Hence, only S_1 passes the test $\lightning x.\checkmark$.

(7.4) Assume $\Gamma = \{(s, \mathtt{r}, y), (s, \mathtt{ns}, y)\}$ and consider

$$S_1 = \Gamma \vdash \langle s \propto \mathtt{r}.x \rangle_z \quad \text{and} \quad S_2 = \Gamma \vdash \langle s \propto \mathtt{ns}.x \rangle_z$$

(note that $S_1 = S_2[^{\mathtt{r}}/_{\mathtt{ns}}]^{\mathrm{i}}$ and $S_2 = S_1[^{\mathtt{ns}}/_{\mathtt{r}}]^{\mathrm{i}}$). After invoking s, S_1 and S_2 respectively reduce to $\Gamma \vdash \langle x \mid y \rangle_z$ and $\Gamma \vdash \langle x \rangle_z \mid y$. Hence, only S_1 passes the test $\lightning y.\overline{z}.\checkmark$ while only S_2 passes $\lightning x.\overline{y}.\overline{z}.\checkmark$.

The inequalities $\mathtt{s} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{ns}$ and $\mathtt{m} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{ns}$ follow by contradiction from $\mathtt{r} =^{\mathrm{i}}_{\mathbf{m}} \mathtt{s} =^{\mathrm{i}}_{\mathbf{m}} \mathtt{m}$ (cf. (23) in Theorem 7.1) and $\mathtt{r} \neq^{\mathrm{i}}_{\mathbf{m}} \mathtt{ns}$.

(7.4) Let $\Gamma = \{(s, \mathtt{rn}, y), (s, \mathtt{n}, y)\}$ and consider

$$S_1 = \Gamma \vdash \langle s \propto \mathtt{rn} \rangle_z \quad \text{and} \quad S_2 = \Gamma \vdash \langle s \propto \mathtt{n} \rangle_z$$

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

(note that $S_1 = S_2[^{\mathtt{rn}}/_{\mathtt{n}}]^{\mathrm{i}}$ and $S_2 = S_1[^{\mathtt{n}}/_{\mathtt{rn}}]^{\mathrm{i}}$). After invoking s, S_1 and S_2 respectively reduce to $\Gamma \vdash \langle y \rangle$ and $\Gamma \vdash z$. Hence, only S_1 passes the test $\overline{y}.\checkmark$, while only S_2 passes $\overline{z}.\checkmark$.

(7.4) Let $\Gamma = \{(s, \mathtt{ns}, y), (s, \mathtt{n}, y)\}$ and consider

$$S_1 = \Gamma \vdash \langle s \propto \mathtt{ns} \rangle_z \quad \text{and} \quad S_2 = \Gamma \vdash \langle s \propto \mathtt{n} \rangle_z$$

(note that $S_1 = S_2[^{\mathtt{ns}}/_{\mathtt{n}}]^{\mathrm{i}}$ and $S_2 = S_1[^{\mathtt{n}}/_{\mathtt{ns}}]^{\mathrm{i}}$). After invoking s, S_1 and S_2 respectively reduce to $\Gamma \vdash y$ and $\Gamma \vdash z$. Hence, only S_1 passes the test $\overline{y}.\checkmark$, while only S_2 passes $\overline{z}.\checkmark$.

8. Conclusions

This paper proposes a semantics of dynamic reconfigurations of transactional scopes in SOC, occurring when service invocations dynamically alter the configuration of transactions possibly introducing new transactional scopes that have to run with the existing ones.

Since both dynamic reconfiguration and LRT are key aspects in SOC, it is crucial to provide a framework to analyse their inter-relationships and to understand and control the mechanisms of failure. The need of a formal semantics for scope reconfiguration is amplified when services support and rely on different kinds of transactional behaviour.

We present a formal model featuring mechanisms to *determine* and *control* the dynamic reconfiguration of distributed transactions. Specifically, 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.

Also, we define a testing framework to study reconfigurable SOC transactions in presence of failures. The proposed framework captures the interplay between the semantics of processes and the dynamic reconfiguration of transactional scopes due to the run-time invocation of new services. Our testing framework allows us to equip EJB-inspired transactions with a suitable notion of *may*- and *must*-order and their induced equivalences. The aim is to test the correctness of the system behaviour, including failure handling and compensations. On this basis, we define notions of testing preorders for ATc systems that show the equivalence of some transactional attributes under suitable conditions. We show that it is possible to replace a transactional attribute with an equivalent one without altering the observable behaviour of the system. Those relations yield a formal framework for comparing transactional attributes. Remarkably, this allows one to specify a larger set of transactional attributes for service invocations. Moreover, our framework has also an impact on modelling aspects of distributed transactions (Bocchi *et al.* 2010).

8.1. *Related work*

The formal investigation of the mechanisms of failure propagation in service orchestration and service-oriented business processes has been addressed by a number of authors.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Process calculi and t ransactions. Many existing approaches extend some process calculi with primitives that allow a party to define transactional scopes, failure handlers, and roll-back or compensation mechanisms. To the best of our knowledge, the formal semantics for SOC transactions in reconfiguring scenarios has not been explicitly addressed; ATc is the first to focus on the interplay between dynamic reconfiguration and failure propagation. Notably Web–π (Laneve and Zavattaro 2005; Mazzara and Lanese 2006) offers a general framework to dynamically relate transactional scopes, however each specific reconfiguration pattern must be explicitly modelled and related to failure propagation. For instance, the Web–π process (where we ignore time for simplicity)

$$\langle\!| \overline{s}(z,x).P\,;\overline{z} |\!\rangle_x \quad | \quad !s(t,u).\langle\!| R;\overline{u} |\!\rangle_t \tag{24}$$

encodes a pattern similar to an ATc transactional invocation with attribute `r` (we can consider the two transactional scopes of (24) as a unique transactional scope since, as illustrated below, the failure of one causes the failure of the other). The leftmost scope x in (24) consists of (*i*) a main process that invokes service s with parameter z and behaves like P, and (*ii*) a compensation process $\overline{z}$. The rightmost process of (24) represents a service provider that upon invocation creates a new instance of scope t with main process R and compensation $\overline{u}$. Upon synchronisation, the parameter t will be replaced by the name of the newly created scope z. The failure of the transactional scope x (resp. z) is triggered by the action $\overline{x}$ (resp. $\overline{z}$). The failure handling of x and z are causally related (since the compensation of x forces the failure z) although the triggering of the failure propagation is not an atomic step as in ATc.

As in ATc, dynamic installation of compensations have been considered in (Guidi *et al.* 2009) and (Vaz *et al.* 2008). Dynamic and static approaches have been compared to failure recovery in (Lanese 2010; Lanese *et al.* 2010). We point out that the primitives for dynamic installation have been proved strictly more expressive than those for static recovery (Lanese *et al.* 2010). A distinguishing feature of ATc with respect to the calculi mentioned above is that dynamic installation is considered in dynamically reconfiguring systems.

The notion of rollback in distributed communications has been studied in (de Vries *et al.* 2010a) in the context of TransCCS, an extension of CCS with primitives for transactional scope definition and error recovery; liveness and safety properties for TransCCS have been studied in (de Vries *et al.* 2010b). Other approaches focus on the notion of compensation. In (Bocchi *et al.* 2003), the $\pi\mathbf{t}$–calculus, an extension of the asynchronous π–calculus inspired by BizTalk, is introduced to model failure handling and compensation in orchestration processes. As $\pi\mathbf{t}$–calculus, StAC (Butler and Ferreira 2004) and CJoin (Bruni *et al.* 2004) are process calculi modelling arbitrarily nested transactions; they focus on the separation of process management with error/compensation. CJoin allows to merge different transactional scopes but does not offer the flexibility of the transactional attributes of ATc. Sagas (Bruni *et al.* 2005) and cCSP (Butler *et al.* 2004) are workflow-based process algebras modelling the occurrence and propagation of compensations in orchestrations with parallel branches.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Transactions and SLAs. The committed cc-pi (Buscemi and Melgratti 2007) enriches the transactional mechanisms of CJoin with primitives for SLA negotiation from cc-pi (Buscemi and Montanari 2007). Committed cc-pi models constraints on non-functional requirements in a more general way with respect to ATc, for example involving more SLA properties, e.g., price of the service, bandwidth. On the other hand, committed cc-pi does not model transactional scope reconfiguration. ATc focuses on one specific SLA property to investigate how it affects the observed behaviour of a system. Also, committed cc-pi models constraint violation, e.g, the binding of two parties with incompatible constraints causes a failure. For simplicity, ATc abstracts from discovery and matchmaking assuming that binding happens only if the properties match and that the environment always offers a suitable match.

8.2. *Future work*

Some of the calculi mentioned above (e.g., $\pi\mathbf{t}$—calculus and sagas) feature a more refined mechanism of commitment with respect to ATc. The mechanism of commit is used to handle the activation of compensations in case of nested transactional scopes. For instance, when the activity in the inner scope terminates without error, or 'commits', its compensation is recorded in the outer scope; in ATc syntax:

$$\langle P \mid \langle Q \rangle_{R_2} \rangle_{R_1} \;\rightarrow^{*}\; \langle P \mid \langle \mathit{Commit} \rangle_{R_2} \rangle_{R_1} \;\rightarrow\; \langle P \rangle_{R_1 | R_2}$$

where *Commit* is the process that has terminated its activity without error. (Note that the relation $\rightarrow$ is not the semantics of ATc and is used only for illustrative purposes.) In this way, an error occurring in the outer scope would also trigger the compensations R_2 of the committed inner scope. In ATc the transactional scopes whose activities have terminated without errors are erased. We illustrate how the example above is expressed in ATc, using $\varnothing$ instead of *Commit* to underline the difference between the semantics of ATc with the notion of commitment presented above. In ATc, the process would evolve as follows:

$$\langle P \mid \langle Q \rangle_{R_2} \rangle_{R_1} \rightarrow^{*} \langle P \mid \langle \varnothing \rangle_{R_2} \rangle_{R_1} \equiv \langle P \rangle_{R_1} \quad \text{by Definition 3.2.}$$

ATc models a simpler semantics for commitment that does not automatically include the compensations of the terminated nested scopes in the compensation of the current scope. Including such mechanisms in ATc, which explicitly models the mechanisms of reconfiguration, would make the calculus (especially the type system) more complex. A higher level notion of commitment which we refer to as 'success' is introduced in Section 5 to characterise, more generally, the correctness of the behaviour of an observed system even in case of failure. We leave the investigation of an encoding of the mechanisms mentioned above in ATc as a future work.

Another limitation of our approach is the lack of link mobility à la π–calculus; extending ATc with name passing 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 framework of ATc will be greatly affected by the introduction of name passing features. Allowing attributes to be communicated is another possible interesting extension of ATc;

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

also, a primitive enabling a service s to make a parameterized invocation to a service s' using the same attribute supported by s (attributes are set when services are published in containers).

An orthogonal topic is the modelling of protocols for deciding the outcome of distributed transactions. Some standards – like Business Transaction Protocol (BTP) (OASIS 2002) and Web Service Transaction (WS-Tx) (OASIS 2009) – 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 (Bocchi 2004; Bocchi and Lucchi 2006) BTP cohesion along with the properties ensured by the 'weakened' constraints have been studied via a formalisation in the asynchronous π–calculus. The interested reader may also refer to (Dalal *et al.* 2003) for an overview on the *cohesion*-base approach of BTP. The present paper provides a high level semantics of failure propagation, compensation and transactional scope reconfiguration, while abstracting from protocols necessary to implement them. Consider, for example, the process $\langle s \propto \mathtt{r}.P \rangle_Q$ 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 transactional scope (if any) of the caller, according to Definition 3.6.

$$\Gamma \vdash \langle s \propto \mathtt{r}.P \rangle_Q \rightsquigarrow^* \Gamma \vdash \langle P \mid P' \rangle_{Q|Q'}$$

The scope on the right-hand side above includes compensations of different possibly cross-domain and distributed processes. Noteworthy, the mechanisms that trigger Q and Q' are not trivial and worth to be investigated. The higher level perspective we adopted has the advantage of providing a concise but rigorous understanding of dynamic reconfigurations of transactional scopes. We leave the investigation of the underneath coordination protocols as a future work.

In Section 7 we presented a number of properties holding for the may-equivalence. We conjecture that similar properties also hold for the must-equivalence. Noticeably, an environment affected by substitution allows more computations than the original one, since each service can be instantiated either in its backward-compatible version or in its substituted one. Whereas, $S \sqsubseteq_{\mathbf{m}} S\sigma$ follows from the fact that if there is a successful computation in S then there is one in $S\sigma$, proving the same property for $\sqsubseteq_{\mathbf{M}}$ would require a different strategy with respect to the one adopted in Section 7 (e.g., to check the substitution does not add diverging computations).

The properties in Figure 2 ensure that successful computations are not 'removed' by attribute substitution. Namely, if S has a successful computation then also $S\sigma$ has a corresponding one. In fact, $S\sigma$ may introduce more possible computations, successful or not, with respect to S. An interesting research direction is to investigate under which conditions, that is under which substitutions σ, $S\sigma \sqsubseteq_{\mathbf{m}} S$ holds. In other words, determine for which attribute substitutions, $S\sigma$ does not have more successful computations than S. We leave the investigation of safety as a future work. Here, we just remark that such investigation would require a new definition of how attribute substitutions are applied to environments. In fact, the current definition of substitution adds extra behaviour to environment since substituted systems correspond to the reuse of the available services in new scenarios.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

As an interesting future direction, we plan to revamp our theory to extend existing semantic models of Java, for instance Featherweight Java (Igarashi *et al.* 1999).

Acknowledgements

We are grateful to Hernan Melgratti and Claudio Mezzina for their valuable comments on the initial phases of this research. Also, we express our gratitude to the anonymous reviewers for their suggestions and constructive criticisms which enabled us to extend our results and significantly improve our paper.

Appendix A. Proofs of Section 4

Propositions A.1 and A.2 below will be tacitly used in the proofs of the lemmas and theorems.

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

Proof. Directly from the definitions. □

Proposition A.2. Operators $_\downarrow_1$, $_\downarrow_2$, and $_\downarrow_3$ distribute over $_ \oplus _$ and $(t_1 \oplus t_2)\downarrow_1 = t_1\downarrow_1 \cup t_2\downarrow_1$.

Proof. Directly from the definitions. □

A.1. *Proof of Lemma 4.1*

The proof of Lemma 4.1 uses the following lemma which proves a similar property for $\textsc{Flatten}(t)$.

Lemma A.1. For any types t and t', $\textsc{Flatten}(t \oplus t') = \textsc{Flatten}(t) \cup \textsc{Flatten}(t')$.

Proof. The proof is by induction on the structure of $t \oplus t'$.
In the base case $t = \varnothing$ (or $t' = \varnothing$) thus $t \oplus t' = t'$ by definition of $\oplus$. Hence

$$\textsc{Flatten}(t \oplus t') = \textsc{Flatten}(t').$$

On the other hand, since $\textsc{Flatten}(t) = \textsc{Flatten}(\varnothing) = \varnothing$ thus

$$\textsc{Flatten}(t) \cup \textsc{Flatten}(t') = \textsc{Flatten}(t').$$

The base case holds since $\textsc{Flatten}(t \oplus t') = \textsc{Flatten}(t) \cup \textsc{Flatten}(t') = \textsc{Flatten}(t')$. The symmetric case ($t' = \varnothing$) is similar.

In the inductive case neither $t = \varnothing$ nor $t' = \varnothing$. Let $t = (I, t_c, t_u)$ and $t' = (I', t'_c, t'_u)$. By definition of $\oplus$, $t \oplus t' = (I \cup I', t_c \oplus t'_c, t_u \oplus t'_u)$. By definition of $\textsc{Flatten}(_)$,

$$\begin{aligned}\textsc{Flatten}(t \oplus t') &= I \cup I' \cup \textsc{Flatten}(t_c \oplus t'_c) \cup \textsc{Flatten}(t_u \oplus t'_u)\\ &= (I \cup \textsc{Flatten}(t_c) \cup \textsc{Flatten}(t_u)) \cup (I' \cup \textsc{Flatten}(t'_c) \cup \textsc{Flatten}(t'_u))\\ &= \textsc{Flatten}(t) \cup \textsc{Flatten}(t')\end{aligned}$$

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

where the last equality holds by inductive hypothesis which guarantees that

$$\textsc{Flatten}(t_c \oplus t'_c) = \textsc{Flatten}(t_c) \cup \textsc{Flatten}(t'_c)$$
$$\textsc{Flatten}(t_u \oplus t'_u) = \textsc{Flatten}(t_u) \cup \textsc{Flatten}(t'_u).$$

□

Next we proof Lemma 4.1. For any types t and t', $\widehat{t \oplus t'} = \widehat{t} \cup \widehat{t'}$.

Proof. We consider two cases, depending on the structure of $t \oplus t'$.

If $t = \varnothing$ (or $t' = \varnothing$), then $t \oplus t' = t'$ by definition of $\oplus$. Hence

$$\widehat{t \oplus t'} = \widehat{t'}.$$

On the other hand, since $\widehat{t} = \widehat{\varnothing} = \varnothing$ thus

$$\widehat{t} \cup \widehat{t'} = \widehat{t'}.$$

This case holds since $\widehat{t \oplus t'} = \widehat{t} \cup \widehat{t'} = \widehat{t'}$. The symmetric case $t' = \varnothing$ is similar.

If $t \neq \varnothing$ and $t' \neq \varnothing$, let $t = (I, t_c, t_u)$ and $t' = (I', t'_c, t'_u)$. By definition of $\oplus$, $t \oplus t' = (I \cup I', t_c \oplus t'_c, t_u \oplus t'_u)$. By definition of $\widehat{\ }$,

$$\widehat{t \oplus t'} = I \cup I' \cup \textsc{Flatten}(t_c \oplus t'_c).$$

On the other hand,

$$\widehat{t} = I \cup \textsc{Flatten}(t_c) \qquad \text{and} \qquad \widehat{t'} = I' \cup \textsc{Flatten}(t'_c)$$

thus

$$\widehat{t} \cup \widehat{t'} = I \cup I' \cup \textsc{Flatten}(t_c) \cup \textsc{Flatten}(t'_c).$$

The thesis is a consequence of Lemma A.1 by which

$$\textsc{Flatten}(t_c \oplus t'_c) = \textsc{Flatten}(t_c) \cup \textsc{Flatten}(t'_c).$$

□

A.2. *Proof of Proposition 4.2*

For any context $\mathbb{C}[\![\natural \fatsemi \boxtimes]\!]$ and $\pi\!:\!Q.P, R, R_0 \in \mathcal{P}$, if $\mathbb{C}[\![\langle \pi\!:\!Q.P \rangle_R \fatsemi R_0]\!] \triangleright t$ then $\mathbb{C}[\![\langle P \rangle_{R|Q} \fatsemi R_0]\!] \triangleright t$.

Proof. If $\mathbb{C}[\![\natural \fatsemi \boxtimes]\!]$ is scope-avoiding, the proof is straightforward noticing that

$$\dfrac{\dfrac{\dfrac{P \triangleright t_p \qquad Q \triangleright t_q}{\pi\!:\!Q.P \triangleright (t_p\downarrow_1, t_p\downarrow_2, t_q \oplus t_p\downarrow_3)}\,\texttt{(comp)} \qquad R \triangleright t_r}{\langle \pi\!:\!Q.P \rangle_R \triangleright ((t_p\downarrow_1 \cup t_p\downarrow_2\downarrow_1)[\![\mathfrak{o} \mapsto \mathfrak{i}]\!], t_p\downarrow_3 \oplus t_p\downarrow_2\downarrow_2 \oplus t_p\downarrow_2\downarrow_3 \oplus t_q \oplus t_r, \varnothing)}\,\texttt{(scope)}}{\dfrac{\dfrac{R \triangleright t_r \qquad Q \triangleright t_q}{R \mid Q \triangleright t_r \oplus t_q}\,\texttt{(par)}}{\langle P \rangle_{R|Q} \triangleright ((t_p\downarrow_1 \cup t_p\downarrow_2\downarrow_1)[\![\mathfrak{o} \mapsto \mathfrak{i}]\!], t_p\downarrow_3 \oplus t_p\downarrow_2\downarrow_2 \oplus t_p\downarrow_2\downarrow_3 \oplus t_q \oplus t_r, \varnothing)}\,\texttt{(scope)}}$$

yields the proof in the base case $\mathbb{C}[\![\natural \fatsemi \boxtimes]\!] = \natural$.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

If $\mathbb{C}[\![\natural \fatsemi \boxtimes]\!]$ is scope-containing, the proof is straightforward by induction on the structure of $\mathbb{C}[\![\natural \fatsemi \boxtimes]\!]$ noticing that, for the base case we have $\mathbb{C}[\![\natural \fatsemi \boxtimes]\!] = \langle \natural \mid P_1 \rangle_{\boxtimes|Q_1}$ and:

$$\dfrac{\dfrac{\dfrac{P \triangleright t_P \qquad R \mid Q \triangleright t_3}{\langle P \rangle_{R|Q} \triangleright t_2}\,(\texttt{scope}) \qquad P_1 \triangleright t_1}{\langle P \rangle_{R|Q} \mid P_1 \triangleright t_2 \oplus t_1}\,(\texttt{par}) \qquad Q_1 \mid R_0 \triangleright t_0}{\langle \langle P \rangle_{R|Q} \mid P_1 \rangle_{Q_1|R_0} \triangleright t'}\,(\texttt{scope})$$

and

$$\dfrac{\dfrac{\dfrac{\dfrac{P \triangleright t_P \qquad Q \triangleright t_Q}{\pi^{\shuffle} QP \triangleright t_4}\,(\texttt{comp}) \qquad R \triangleright t_R}{\langle \pi^{\shuffle} QP \rangle_R \triangleright t_2'}\,(\texttt{scope}) \qquad P_1 \triangleright t_1}{\langle \pi^{\shuffle} QP \rangle_R \mid P_1 \triangleright t_2' \oplus t_1}\,(\texttt{scope}) \qquad Q_1 \mid R_0 \triangleright t_0}{\langle \langle \pi^{\shuffle} QP \rangle_R \mid P_1 \rangle_{Q_1|R_0} \triangleright t}\,(\texttt{scope})$$

therefore, proving that $t_2' = t_2$ we have $t = t'$. For this, first notice that $t_3 = t_Q \oplus t_R$. Also, by definition of `(scope)` and `(icomp)`, we have

$$t_2 = ((t_P\downarrow_1 \cup t_P\downarrow_2\downarrow_1)[\![\mathfrak{o} \mapsto \mathfrak{i}]\!], t_P\downarrow_3 \oplus t_P\downarrow_2\downarrow_2 \oplus t_P\downarrow_2\downarrow_3 \oplus t_3, \varnothing)$$
$$t_4 = (t_P\downarrow_1, t_P\downarrow_2, t_Q \oplus t_P\downarrow_3)$$

which, again by definition of `(scope)`, yields $t_2' = t_2$.

The case $\mathbb{C}[\![\natural \fatsemi \boxtimes]\!] = \langle \mathbf{C}[\![\natural \fatsemi \boxtimes]\!] \rangle_{Q_1}$ easily follows by induction. □

A.3. *Proof of Theorem 4.2*

Let the environment Γ and $P \in \mathcal{P}$ be well-typed. If $\Gamma \vdash P \rightsquigarrow \Gamma \vdash Q$ then Q is well-typed.

Proof. The proof is by case analysis on the reduction $\Gamma \vdash P \rightsquigarrow \Gamma \vdash Q$ (cf. Definition 3.6). Hereafter, we assume $P \triangleright t$ and $Q \triangleright t'$.

`(s1)` The thesis follows directly from Theorem 4.1.

`(s2)` This case does not apply because otherwise $(\mathfrak{o},\mathtt{m}) \in \widehat{t}$ with $\Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![s \propto A.P]\!] \triangleright t$, contrary to the well-typedness hypothesis.

`(s3)` Assume $P = \mathbf{A}[\![s \propto A.P_1]\!]$ for a scope-avoiding context $\mathbf{A}[\![\natural]\!]$ and $R \in \Gamma(s, \{\mathtt{s},\mathtt{n},\mathtt{ns}\} \cap A)$ such that $\Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![s \propto A.P_1]\!] \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ (\mathbf{A}[\![P_1]\!]) \mid R$. Let $\mathbf{A}[\![P_1]\!] \triangleright t_1$ and $R \triangleright t_2$, then $(\mathfrak{o},\mathtt{m}) \notin \widehat{t_1}$ (by well-typedness of P and Definition 4.4) and $(\mathfrak{o},\mathtt{m}) \notin \widehat{t_2}$ (by well-typedness of Γ and Definition 4.5); therefore $(\mathfrak{o},\mathtt{m}) \notin \widehat{t_1 \oplus t_2}$ (which gives the thesis by Lemma 4.1).

`(s4)` This case is similar to the previous one.

`(s5)` Let P be of the form $\mathbf{C}[\![s \propto A.P_1 \fatsemi P_2]\!]$ and $R \in \Gamma(s, \{\mathtt{m},\mathtt{s},\mathtt{r}\} \cap A)$ such that $\Gamma \vdash \nu\tilde{x}\ P \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ \mathbf{C}[\![P_1 \mid R \fatsemi P_2]\!]$. If $R \triangleright t_R$, then $t' = t'' \oplus t_R$ for a type $\widehat{t''} \subseteq \widehat{t}$ (i.e., $\widehat{t''}$ may not include $(\mathfrak{i},a)$ for some $a \in A$). If $(\mathfrak{o},\mathtt{m}) \in \widehat{t'}$ then it must be $(\mathfrak{o},\mathtt{m}) \in \widehat{t_R}$ (otherwise it

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

would be $(\mathtt{o},\mathtt{m}) \in \widehat{t''}$ therefore $(\mathtt{o},\mathtt{m}) \in \widehat{t}$ since $\widehat{t''} \subseteq \widehat{t}$). This would yield a contradiction by the hypothesis that Γ is well-typed.

(s6) If $P = \mathbb{C}[\![\mathbf{C}[\![s \propto A.P_1 \fatsemi P_2]\!] \fatsemi P_3]\!]$ with $\mathtt{n} \in A$ then the thesis follows by observing that the type of $\mathbb{C}[\![P_2 \fatsemi P_3]\!] \triangleright t_Q$ with $\widehat{t_Q} \subseteq \widehat{t}$.

(s7), (s8) This cases are similar to the case for (s5).

□

Appendix B. Proofs of Section 6

The results proved below can be generalised to take into account name restriction. This would just make the proofs more complex without bringing any advantage.

B.1. *Proof of Proposition 6.1*

Given an observed system $\Gamma \vdash \langle P \rangle_Q \mid R \parallel O$, if

$$\Gamma \vdash \langle P \rangle_Q \mid R \parallel O \quad \rightsquigarrow \quad S \parallel O' \tag{25}$$

with $S \parallel O'$ non-diverging, then $S = \Gamma \vdash \langle P' \rangle_Q \mid R'$ or $S = \Gamma \vdash Q \mid R$.

Proof. The proof is by induction on the derivation of (25) considering that axiom (os3) cannot be applied otherwise $S \parallel O'$ would be diverging.

To apply (os1), O must either synchronise with P or with R. If O synchronises with R the thesis follows trivially; in the other case P has the form $\mathbb{C}[\![\pi \fatbar Q'.P' \fatsemi R_0]\!]$ hence $S = \Gamma \vdash \langle \mathbb{C}[\![P' \fatsemi R_0 \mid Q']\!] \rangle_Q \mid R$.

If (25) is an instance of (os2) then $O = \lightning\pi.O'$ and it can let fail either an action of P or one of R. In the latter case the thesis follows trivially, while in the former case we have $P = \mathbb{C}[\![\langle \pi \fatbar Q' \mid P_1' \rangle_Q.P_1 \fatsemi P_2]\!]$, hence $S = \Gamma \vdash Q \mid R$ if $\mathbb{C}[\![\square \fatsemi \boxtimes]\!]$ is scope-avoiding, otherwise $S = \Gamma \vdash \langle P_2 \rangle_Q \mid R$.

If (25) is obtained by (os4) then $\Gamma \vdash \langle P \rangle_Q \mid R \parallel O$ progresses either (*i*) because P and R communicate with each other or (*ii*) because one of them invokes a service in Γ. Observing that rule (s2) (cf. Definition 3.6) cannot be used to derive (25) because, by hypothesis, $S \parallel O$ is non-diverging, we conclude that in both cases (*i*) and (*ii*) that S has the form $\Gamma \vdash \langle P' \rangle_Q \mid R'$ for suitable P' and R'.

If (25) is obtained by applying rules (os5) or (os6) then the thesis follows from the inductive hypothesis. □

B.2. *Proof of Lemma 6.1*

Let $\Gamma \vdash P$ be a prudent system. For any observer O and any $R \in \mathcal{P}$

$$\Gamma \vdash P \mid R \parallel O \rightsquigarrow \Gamma \vdash P' \mid R' \parallel O' \quad \Longrightarrow \quad \Gamma \vdash \langle P \rangle_Q \mid R \parallel O \rightsquigarrow S \parallel O' \tag{26}$$

where $S = \Gamma \vdash \langle P' \rangle_Q \mid R''$ for some $R'' \in \mathcal{P}$ or $S = \Gamma \vdash Q \mid R'$.

Proof. Call $\mathfrak{t}$ the transition of the hypothesis of the implication (26). The proof is by induction on the derivation of $\mathfrak{t}$.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

The axioms in Definition 5.2 can be applied to let O interact with either P or R. In the latter case it is easy to verify that $\Gamma \vdash \langle P \rangle_Q \mid R \parallel O \rightsquigarrow \Gamma \vdash \langle P \rangle_Q \mid R' \parallel O'$, hence we focus on the former case. First consider that $\mathfrak{t}$ is derived from the axioms in Definition 5.2.

— If $\mathfrak{t}$ is an instance of `(os1)`, $O = \overline{\pi}.O'$ for an observer O' and $P = \mathbb{C}[\![\pi \uplus Q'.P_1 \fatsemi R_0]\!]$ for a context $\mathbb{C}[\![\square \fatsemi \boxtimes]\!]$ and $P_1, Q' \in \mathcal{P}$; hence $\Gamma \vdash \langle P \rangle_Q \mid R \parallel O \rightsquigarrow \Gamma \vdash \langle \mathbb{C}[\![P_1 \fatsemi R_0 \mid Q']\!] \rangle_Q \mid R \parallel O'$, by `(os1)`.

— If $\mathfrak{t}$ is an instance of `(os2)` then $O = \not{\pi}.O'$ for an observer O' and

  – either $P = \mathbf{A}[\![\pi.P_1]\!]$ for a scope-avoiding context $\mathbf{A}[\![\square]\!]$ and $P_1 \in \mathcal{P}$; then, by `(os2)`, $\Gamma \vdash \langle P \rangle_Q \mid R \rightsquigarrow \Gamma \vdash Q \mid R \parallel O'$;

  – or $P = \mathbb{C}[\![\langle \pi \uplus Q'.P_1 \mid R' \rangle_{P_1'} \fatsemi P_2]\!]$ for a context $\mathbb{C}[\![\square \fatsemi \boxtimes]\!]$ and $P_1, P_1', R', Q', P_2 \in \mathcal{P}$; then

$$\Gamma \vdash \langle \mathbb{C}[\![\langle \pi \uplus Q'.P_1 \mid R' \rangle_{P_1'} \fatsemi P_2]\!] \rangle_Q \mid R \parallel O \rightsquigarrow \Gamma \vdash \langle \mathbb{C}[\![P_1' \fatsemi P_2]\!] \rangle_Q \mid R \parallel O'$$

  by `(os2)`.

— If $\mathfrak{t}$ is an instance of `(os3)` then $O = \not{\pi}.O'$ for an observer O' and $P = \mathbf{A}[\![\pi \uplus Q'.P_1]\!]$ for a scope-avoiding context $\mathbf{A}[\![\square]\!]$ and $P_1, Q' \in \mathcal{P}$; hence, $\langle P \rangle_Q \equiv \mathbf{C}[\![\pi \uplus Q'.P_1 \fatsemi Q]\!]$ for a scope-containing $\mathbf{C}[\![\square \fatsemi \boxtimes]\!]$; hence, by `(os2)` and `(os6)`, $\Gamma \vdash \langle P \rangle_Q \mid R \rightsquigarrow \Gamma \vdash Q \mid R \parallel O'$.

If $\mathfrak{t}$ is obtained with a derivation whose last rule is `(os4)`, the thesis follows by inspection of the rules for system reduction in Definition 3.6 noticing that rules `(s3)` and `(s4)` cannot be used to derive $\mathfrak{t}$ as $\Gamma \vdash P$ is prudent. (Note that `(s2)` cannot be used as P is prudent, hence well-typed.)

If the last step in the derivation of $\mathfrak{t}$ is an application of `(os5)` or `(os6)`, the thesis follows from the inductive hypothesis. □

Appendix C. Proofs of Section 7

It is convenient to generalise attribute substitutions to contexts. Given an attribute substitution σ, we call $\square\sigma$ and $\boxtimes\sigma$ *generalised holes*. A *generalised context* is a context (as defined in Definition 3.3 on page 626) with generalised holes.

Remark C.1. Notice that contexts as in Definition 3.3 are generalised contexts where all substitutions are identities (i.e., $[^a/_a]$ for an $a \in \mathcal{A}$).

When applying substitutions to contexts we obtain generalised contexts according to the following definition.

Definition C.1 (attribute substitution on contexts). Let σ be an attribute substitution. Define, for any scope-avoiding context $\mathbf{A}[\![\square]\!]$

$$\mathbf{A}[\![\square]\!]\sigma \stackrel{def}{=} \begin{cases} \square\sigma, & \text{if } \mathbf{A}[\![\square]\!] = \square \\ P\sigma \mid \square\sigma, & \text{if } \mathbf{A}[\![\square]\!] = P \mid \square \\ \square\sigma \mid P\sigma, & \text{if } \mathbf{A}[\![\square]\!] = \square \mid P \end{cases}$$

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

and for any scope-containing context $\mathbf{C}[\![\natural \ ; \ \boxtimes]\!]$

$$\mathbf{C}[\![\natural \ ; \ \boxtimes]\!]\sigma \stackrel{def}{=} \begin{cases} \langle \mathbf{C}[\![\natural \ ; \ \boxtimes]\!]\rangle_{Q\sigma}, & \text{if } \mathbf{C}[\![\natural \ ; \ \boxtimes]\!] = \langle \mathbf{C}[\![\natural \ ; \ \boxtimes]\!]\rangle_{Q} \wedge \sigma = [^{b}/_{a}]^{\mathtt{o}} \\ \langle \natural \mid P\rangle_{Q\sigma|\boxtimes\sigma}, & \text{if } \mathbf{C}[\![\natural \ ; \ \boxtimes]\!] = \langle \natural \mid P\rangle_{Q|\boxtimes} \wedge \sigma = [^{b}/_{a}]^{\mathtt{o}} \\ \langle \mathbf{C}[\![\natural \ ; \ \boxtimes]\!][^{b}/_{a}]\rangle_{Q\sigma}, & \text{if } \mathbf{C}[\![\natural \ ; \ \boxtimes]\!] = \langle \mathbf{C}[\![\natural \ ; \ \boxtimes]\!]\rangle_{Q} \wedge (\sigma = [^{b}/_{a}]^{\mathtt{i}} \vee \sigma = [^{b}/_{a}]) \\ \langle \natural[^{b}/_{a}] \mid P[^{b}/_{a}]\rangle_{Q\sigma|\boxtimes\sigma}, & \text{if } \mathbf{C}[\![\natural \ ; \ \boxtimes]\!] = \langle \natural \mid P\rangle_{Q|\boxtimes} \wedge (\sigma = [^{b}/_{a}]^{\mathtt{i}} \vee \sigma = [^{b}/_{a}]). \end{cases}$$

Finally, define $\mathbb{C}[\![\natural \ ; \ \boxtimes]\!]\sigma$ as $\mathbf{A}[\![\natural]\!]\sigma$ if $\mathbb{C}[\![\natural \ ; \ \boxtimes]\!] = \mathbf{A}[\![\natural]\!]$ or as $\mathbf{C}[\![\natural \ ; \ \boxtimes]\!]\sigma$ if $\mathbb{C}[\![\natural \ ; \ \boxtimes]\!] = \mathbf{C}[\![\natural \ ; \ \boxtimes]\!]$.

Remark C.2. The substitution action on contexts does not affect their structure as, by Definition C.1, in $\mathbb{C}[\![\natural \ ; \ \boxtimes]\!]\sigma$ the substitution σ percolates parallel composition and scopes.

C.1. *Proof of Proposition 7.3*

Let S be a prudent system such that $S \rightsquigarrow S'$ for a system S'. If $\sigma \in \{[^{\mathtt{rn}}/_{\mathtt{ns}}]^{\mathtt{o}}\} \cup \{[^{\mathtt{b}}/_{\mathtt{a}}]^{\mathtt{o}} \mid a \in \{\mathtt{m},\mathtt{n},\mathtt{r}\} \wedge b \in \mathcal{A}\} \cup \{[^{\mathtt{b}}/_{\mathtt{a}}]^{\mathtt{o}} \mid a,b \in \{\mathtt{n},\mathtt{s},\mathtt{ns}\}\}$ then there is S'' such that

$$S\sigma \rightsquigarrow S'' \quad \text{and} \quad S'\sigma \sqsubseteq_{\mathbf{m}} S''. \tag{27}$$

Proof. In the proof it is convenient to denote $\mathbb{C}[\![\natural \ ; \ \boxtimes]\!]\sigma$ as $\mathbb{C}_{\sigma}[\![\natural \ ; \ \boxtimes]\!]$.

We proceed by case analysis on the derivation of $S \rightsquigarrow S'$ as per Definition 3.6 (page 629). Below, for the cases $[^{b}/_{a}]^{\mathtt{o}}$ with $a = b \in \{\mathtt{n},\mathtt{s},\mathtt{ns}\}$, we tacitly use Lemma 7.1 which trivially implies the thesis. For the cases $[^{b}/_{a}]^{\mathtt{o}}$ with $a \in \{\mathtt{m},\mathtt{n},\mathtt{r}\}$ and $b \in \mathcal{A}$ the thesis follows from the fact that, since S is prudent then $S = S\sigma$ for these substitutions.

Case (s1)**.** In this case $S = \Gamma \vdash P$ for a process P and $P \to P'$. By Proposition 7.2, $P\sigma \to P'\sigma$ which gives the thesis (since $S'' = S'\sigma$).

Case (s2)**.** This case does not apply since S is prudent (hence well-typed).

Case (s3)**.** As in the previous case, $S = \Gamma \vdash v\tilde{x}\ \mathbf{A}[\![s \propto A.P]\!]$ and there is $R \in \Gamma(s,\{\mathtt{s},\mathtt{n},\mathtt{ns}\} \cap A)$ so that $S \rightsquigarrow S' = \Gamma \vdash v\tilde{x}\ \mathbf{A}[\![P]\!] \mid R$ by (s3).

If $\sigma = [^{\mathtt{rn}}/_{\mathtt{ns}}]^{\mathtt{o}}$ then $S\sigma = \Gamma\sigma \vdash v\tilde{x}\ \mathbf{A}_{\sigma}[\![s \propto \sigma A.P\sigma]\!]$. Assume first $\{\mathtt{s},\mathtt{n}\} \cap A \neq \varnothing$ and $R \in \Gamma(s,\{\mathtt{s},\mathtt{n}\})$, then also $\{\mathtt{s},\mathtt{n}\} \cap \sigma A \neq \varnothing$ and $R\sigma \in \Gamma\sigma(s,\{\mathtt{s},\mathtt{n}\})$, hence $S\sigma \rightsquigarrow \Gamma\sigma \vdash v\tilde{x}\ \mathbf{A}_{\sigma}[\![P\sigma]\!] \mid R\sigma = S'\sigma$ by (s3).

Otherwise, by definition $R\sigma \in \Gamma\sigma(s,\sigma(\mathtt{ns}))$ with $\sigma(\mathtt{ns}) = \mathtt{rn}$ and $S\sigma \rightsquigarrow \Gamma\sigma \vdash v\tilde{x}\ \mathbf{A}_{\sigma}[\![P\sigma]\!] \mid \langle R\sigma\rangle = S''$ by (s4), hence $S'\sigma \sqsubseteq_{\mathbf{m}} S''$ by Theorem 6.1.

The cases in which $\mathtt{r}$ or $\mathtt{m}$ is the attribute to be substituted cannot occur since S is prudent. If $[^{\mathtt{b}}/_{\mathtt{a}}]^{\mathtt{o}}$ with $a \neq b \in \{\mathtt{n},\mathtt{s},\mathtt{ns}\}$, $R\sigma \in \Gamma\sigma(s,\{\sigma(\mathtt{n}),\sigma(\mathtt{s}),\sigma(\mathtt{ns})\} \cap \sigma A)$ holds by construction, therefore $S\sigma \rightsquigarrow S'\sigma = v\tilde{x}\ \mathbf{A}_{\sigma}[\![P\sigma]\!] \mid R\sigma$ by (s3).

Case (s4)**.** In this case $S = \Gamma \vdash v\tilde{x}\ \mathbf{A}[\![s \propto A.P]\!]$ and there is $R \in \Gamma(s,\{\mathtt{r},\mathtt{rn}\} \cap A)$ so that $S \rightsquigarrow S' = \Gamma \vdash v\tilde{x}\ \mathbf{A}[\![P]\!] \mid \langle R\rangle$ by (s4). Note that $\mathtt{r} \notin A$ (since S is prudent), that $\mathtt{rn} \in \sigma A$, and that $S\sigma = \Gamma\sigma \vdash v\tilde{x}\ \mathbf{A}_{\sigma}[\![s \propto \sigma A.P\sigma]\!]$ by definition. Therefore

$$S\sigma \rightsquigarrow \Gamma\sigma \vdash v\tilde{x}\ \mathbf{A}_{\sigma}[\![P\sigma]\!] \mid \langle R\rangle = S'\sigma$$

is obtained by (s4) since $R \in \Gamma\sigma(s,\{\mathtt{rn}\} \cap \sigma A)$ by construction.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Case (s5). In this case, $S = \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\mathbf{C}[\![s \propto A.P \ ; \ Q]\!] \ ; \ Q']\!]$ and there is $R \in \Gamma(s, \{\mathtt{m}, \mathtt{s}, \mathtt{r}\} \cap A)$ such that $\mathrm{fc}(R) \cap \tilde{x} = \varnothing$ and $S \rightsquigarrow S' = \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\mathbf{C}[\![P \mid R \ ; \ Q]\!] \ ; \ Q']\!]$. By definition $S\sigma = \Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}_\sigma[\![\mathbf{C}[\![s \propto A.P \ ; \ Q]\!] \ ; \ Q']\!]$. Hence, by (s5)

$$S\sigma \quad \rightsquigarrow \quad \Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}_\sigma[\![\mathbf{C}[\![P \mid R \ ; \ Q]\!] \ ; \ Q']\!] = S'\sigma$$

since $R \in \Gamma\sigma(s, \{\mathtt{m}, \mathtt{s}, \mathtt{r}\} \cap A)$ by construction.

Case (s6). Given $S = \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\langle s \propto A.P \mid P' \rangle_Q \ ; \ Q']\!]$ with $\mathtt{n} \in A$, we have $S \rightsquigarrow S' = \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![Q \ ; \ Q']\!]$. Also, $S'\sigma = \Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}_\sigma[\![Q \ ; \ Q']\!]$. Then

$$S\sigma = \Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}_\sigma[\![\langle s \propto A.P \mid P' \rangle_Q \ ; \ Q']\!] \quad \rightsquigarrow \quad \Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}_\sigma[\![Q \ ; \ Q']\!] = S'\sigma$$

by rule (s6) since $\mathtt{n} \in A$.

Case (s7). In this case $S = \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\mathbf{C}[\![s \propto A.P \ ; \ Q]\!] \ ; \ Q']\!]$ and there is $R \in \Gamma(s, \{\mathtt{ns}\} \cap A)$ such that $S \rightsquigarrow S' = \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\mathbf{C}[\![P \ ; \ Q]\!] \ ; \ Q']\!] \mid R$. Similarly to the previous case, $S'\sigma = \Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}_\sigma[\![\mathbf{C}[\![P \ ; \ Q]\!] \ ; \ Q']\!] \mid R\sigma$. By (s7),

$$S\sigma = \Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}_\sigma[\![\mathbf{C}[\![s \propto A.P \ ; \ Q]\!] \ ; \ Q']\!] \rightsquigarrow \Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}_\sigma[\![\mathbf{C}[\![P \ ; \ Q]\!] \ ; \ Q']\!] \mid R\sigma = S'\sigma$$

since $R\sigma \in \Gamma\sigma(s, \{\mathtt{ns}\} \cap A)$ by construction.

Case (s8). In this case, $S = \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\mathbf{C}[\![s \propto A.P \ ; \ Q]\!] \ ; \ Q']\!] \rightsquigarrow S' = \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\mathbf{C}[\![P \ ; \ Q]\!] \ ; \ Q']\!] \mid \langle R \rangle$ for an $R \in \Gamma(s, \{\mathtt{rn}\} \cap A)$. Therefore, $S'\sigma = \Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}_\sigma[\![\mathbf{C}[\![P \ ; \ Q]\!] \ ; \ Q']\!] \mid \langle R \rangle$. Since by definition $R \in \Gamma\sigma(s, \{\mathtt{rn}\} \cap A)$, by (s8)

$$S\sigma = \Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}_\sigma[\![\mathbf{C}[\![s \propto A.P \ ; \ Q]\!] \ ; \ Q']\!] \rightsquigarrow \Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}_\sigma[\![\mathbf{C}[\![P \ ; \ Q]\!] \ ; \ Q']\!] \mid \langle R \rangle$$

which yields the thesis.

□

C.2. *Proof of Proposition 7.4*

Let σ be either of the substitutions $[^{\mathtt{rn}}/_{\mathtt{ns}}]^{\mathrm{i}}$ or $[^{b}/_{a}]^{\mathrm{i}}$ with $a, b \in \{\mathtt{r}, \mathtt{s}, \mathtt{m}\}$, then

$$S \rightsquigarrow S' \implies S\sigma \rightsquigarrow S'' \text{ with } S'\sigma \sqsubseteq_{\mathtt{m}} S''$$

for any system S.

Proof. The proof proceeds by case analysis on the derivation of $S \rightsquigarrow S'$ as per Definition 3.6 (page 629).

Case (s1). Similarly to the case (s1) in the proof of Proposition 7.3.

Case (s2). In this case $S = \Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![s \propto A.P]\!] \rightsquigarrow S' = \mathbf{A}[\![\mathtt{err}]\!]$ with $\mathtt{m} \in A$; hence, $S\sigma = \Gamma\sigma \vdash \nu\tilde{x}\ \mathbf{A}_\sigma[\![s \propto A.P]\!]$, therefore $S\sigma \rightsquigarrow \Gamma\sigma \vdash \nu\tilde{x}\ \mathbf{A}_\sigma[\![\mathtt{err}]\!] = S'\sigma$ by (s2).

Case (s3). As in the previous case, $S = \Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![s \propto A.P]\!] \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![P]\!] \mid R = S'$ for an $R \in \Gamma(s, \{\mathtt{s}, \mathtt{n}, \mathtt{ns}\} \cap A)$; hence $S\sigma = \Gamma\sigma \vdash \nu\tilde{x}\ \mathbf{A}_\sigma[\![s \propto A.P]\!]$ and $R\sigma \in \Gamma\sigma(s, \{\mathtt{s}, \mathtt{n}, \mathtt{ns}\} \cap A)$. Therefore, $S\sigma \rightsquigarrow \Gamma\sigma \vdash \nu\tilde{x}\ \mathbf{A}_\sigma[\![P]\!] \mid R\sigma = S'\sigma$ by (s3).

Case (s4). We have $S = \Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![s \propto A.P]\!] \rightsquigarrow S' = \Gamma \vdash \nu\tilde{x}\ \mathbf{A}[\![P]\!] \mid \langle R \rangle$ where $R \in \Gamma(s, \{\mathtt{r}, \mathtt{rn}\} \cap A)$. By (s4)

$$S\sigma = \Gamma\sigma \vdash \nu\tilde{x}\ \mathbf{A}_\sigma[\![s \propto A.P]\!] \quad \rightsquigarrow \quad \Gamma\sigma \vdash \nu\tilde{x}\ \mathbf{A}_\sigma[\![P]\!] \mid \langle R\sigma \rangle = S'\sigma$$

since $R\sigma \in \Gamma\sigma(s, \{\mathtt{r}, \mathtt{rn}\} \cap A)$ by construction.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Case (s5). In this case, $S = \Gamma \vdash \nu\tilde{x}\, \mathbb{C}[\![\mathbf{C}[\![s \propto A.P \;;\; Q]\!] \;;\; Q']\!]$ and for an $R \in \Gamma(s, \{\mathtt{m},\mathtt{s},\mathtt{r}\} \cap A)$ such that $\mathsf{fc}(R) \cap \tilde{x} = \varnothing$, $S \rightsquigarrow S' = \Gamma \vdash \nu\tilde{x}\, \mathbb{C}[\![\mathbf{C}[\![P \mid R \;;\; Q]\!] \;;\; Q']\!]$. Notice that $S'\sigma = \nu\tilde{x}\, \mathbb{C}_\sigma[\![\mathbf{C}[\![P\sigma \mid R\sigma \;;\; Q]\!] \;;\; Q']\!]$. Since $\{\mathtt{m},\mathtt{s},\mathtt{r}\} \cap \sigma A \neq \varnothing$, by (s5) and Definition 7.1, we have

$$S\sigma = \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\mathbf{C}[\![s \propto \sigma A.P\sigma \;;\; Q]\!] \;;\; Q']\!] \quad \rightsquigarrow \quad \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\mathbf{C}[\![P\sigma \mid R\sigma \;;\; Q]\!] \;;\; Q']\!]$$

since $R\sigma \in \Gamma\sigma(s, \{\mathtt{m},\mathtt{s},\mathtt{r}\} \cap \sigma A)$; hence $\Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\mathbf{C}[\![P\sigma \mid R\sigma \;;\; Q]\!] \;;\; Q']\!] = S'\sigma$.

Case (s6). Assume $S = \Gamma \vdash \nu\tilde{x}\, \mathbb{C}[\![\langle s \propto A.P \mid P' \rangle_Q \;;\; Q']\!]$ with $\mathtt{n} \in A$. We have $S \rightsquigarrow S' = \Gamma \vdash \mathbb{C}[\![Q \;;\; Q']\!]$. Then $S'\sigma = \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![Q \;;\; Q']\!]$. By (s6) and the Definition 7.1,

$$S\sigma = \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\langle s \propto A.P \mid P' \rangle_Q \;;\; Q']\!] \quad \rightsquigarrow \quad \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![Q \;;\; Q']\!]$$

since $\mathtt{n} \in \sigma A$, therefore $\Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}[\![Q \;;\; Q']\!] = S'\sigma$.

Case (s7). We have $S = \Gamma \vdash \nu\tilde{x}\, \mathbb{C}[\![\mathbf{C}[\![s \propto A.P \;;\; Q]\!] \;;\; Q']\!] \rightsquigarrow S' = \Gamma \vdash \nu\tilde{x}\, \mathbb{C}[\![\mathbf{C}[\![P \;;\; Q]\!] \;;\; Q']\!] \mid R$ by (s7) for an $R \in \Gamma(s, \{\mathtt{ns}\} \cap A)$. Then $S'\sigma = \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\mathbf{C}[\![P \;;\; Q]\!] \;;\; Q']\!] \mid R\sigma$. If $\sigma = [^{\mathtt{rn}}/_{\mathtt{ns}}]^{\mathrm{i}}$ then $\sigma A = A \setminus \{\mathtt{ns}\} \cup \{\mathtt{rn}\}$ and, by definition, $R\sigma \in \Gamma\sigma(s, \{\mathtt{rn}\} \cap \sigma A)$, therefore, by (s8)

$$S\sigma = \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\mathbf{C}[\![s \propto A.P \;;\; Q]\!] \;;\; Q']\!] \;\rightsquigarrow\; \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\mathbf{C}[\![P \;;\; Q]\!] \;;\; Q']\!] \mid \langle R\sigma \rangle.$$

Hence, $S'\sigma \sqsubseteq_{\mathbf{m}} \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\mathbf{C}[\![P \;;\; Q]\!] \;;\; Q']\!] \mid \langle R\sigma \rangle$ by Theorem 6.1.

For any σ such that ns is not in the domain of σ, $\mathtt{ns} \in \sigma A$ and $R\sigma \in \Gamma\sigma(s, \mathtt{ns})$ by construction; hence $S\sigma \rightsquigarrow \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\mathbf{C}[\![P \;;\; Q]\!] \;;\; Q']\!] \mid R\sigma$ by (s7).

Case (s8). We have $S = \Gamma \vdash \nu\tilde{x}\, \mathbb{C}[\![\mathbf{C}[\![s \propto A.P \;;\; Q]\!] \;;\; Q']\!] \rightsquigarrow S' = \Gamma \vdash \nu\tilde{x}\, \mathbb{C}[\![\mathbf{C}[\![P \;;\; Q]\!] \;;\; Q']\!] \mid \langle R \rangle$ for an $R \in \Gamma(s, \{\mathtt{rn}\} \cap A)$. By definition, $S'\sigma = \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\mathbf{C}[\![P \;;\; Q]\!] \;;\; Q']\!] \mid \langle R\sigma \rangle$. Since $R\sigma \in \Gamma\sigma(s, \{\mathtt{rn}\} \cap \sigma A)$ by construction,

$$S\sigma = \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\mathbf{C}[\![s \propto A.P \;;\; Q]\!] \;;\; Q']\!] \rightsquigarrow \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\mathbf{C}[\![P \;;\; Q]\!] \;;\; Q']\!] \mid \langle R\sigma \rangle$$

by (s8); hence $\Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\mathbf{C}[\![P \;;\; Q]\!] \;;\; Q']\!] \mid \langle R\sigma \rangle = S'\sigma$.

□

C.3. *Proof of Lemma 7.2*

Let $\mathfrak{t}$ be a transition $S \parallel O \rightsquigarrow S' \parallel O'$ of an observed system $S \parallel O$. Then

$$(S \parallel O)[^b/_a]^{\mathrm{o}} \rightsquigarrow (S' \parallel O')[^b/_a]^{\mathrm{o}} \qquad \text{if } a, b \in \{\mathtt{n}, \mathtt{ns}, \mathtt{s}\} \tag{28}$$

$$(S \parallel O)[^b/_a]^{\mathrm{i}} \rightsquigarrow (S' \parallel O')[^b/_a]^{\mathrm{i}} \qquad \text{if } a, b \in \{\mathtt{r}, \mathtt{s}, \mathtt{m}\}. \tag{29}$$

Namely, $\mathfrak{t}$ is preserved by the substitutions specified in (28) and (29).

Proof. The proof is by induction on the derivation of t according to Definition 5.2 (on page 639). Hereafter, $\sigma \in \{[^b/_a]^{\mathrm{o}}, [^b/_a]^{\mathrm{i}}\}$ (recall that $(S \parallel O)\sigma = S\sigma \parallel O$ by Definition 7.2).

If $\mathfrak{t}$ is an instance of (os1) then $S \parallel O$ is of the form $\Gamma \vdash \nu\tilde{x}\, \mathbb{C}[\![\pi.P \;;\; R]\!] \parallel \overline{\pi}.O$ (with $\pi, \overline{\pi} \cap \tilde{x} = \varnothing$) and $S' \parallel O'$ of the form $\Gamma \vdash \nu\tilde{x}\, \mathbb{C}[\![P \;;\; R \mid Q]\!] \parallel O'$. Then $S\sigma = \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![\pi.\hat{P} \;;\; R]\!]$. Therefore

$$S\sigma \parallel O \quad \rightsquigarrow \quad \Gamma\sigma \vdash \nu\tilde{x}\, \mathbb{C}_\sigma[\![P \;;\; Q]\!] \parallel O'.$$

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Hence, $\Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}'[\![P \fatsemi Q]\!] \parallel O' = S'\sigma \parallel O'$.

If $\mathfrak{t}$ is an instance of `(os2)` or `(os3)` the proof is similar to the previous case.

If $\mathfrak{t}$ is an instance of `(os4)` then, by Corollaries 7.1 and 7.3, $S\sigma \rightsquigarrow S'\sigma$, therefore $S\sigma \parallel O \rightsquigarrow S'\sigma \parallel O$ by `(os4)`.

If $\mathfrak{t}$ is obtained with a proof ending with an application of `(os5)` or `(os6)`, the thesis trivially follows by induction. □

C.4. *Proof of Theorem 7.2*

Consider the order relations

(a) $\mathtt{ns} \leqslant^{\mathtt{o}}_{\mathbf{m}} \mathtt{rn}$ (b) $\mathtt{m} \leqslant^{\mathtt{o}}_{\mathbf{m}} a,\ a \in \mathcal{A}$

(c) $a \leqslant^{\mathtt{o}}_{\mathbf{m}} b,\ a \in \{\mathtt{n},\mathtt{r}\}, b \in \mathcal{A}$ (d) $\mathtt{ns} \leqslant^{\mathtt{i}}_{\mathbf{m}} \mathtt{rn}$

We have that (a), (c) and (d) hold for prudent systems, while (b) holds for any system.

Proof. Let S be a prudent system and σ one of the substitutions in (a), (c) or (d), or let S be a (prudent or non-prudent) system and σ one of the substitutions in (b). Call $\mathfrak{t}$ the transition $S \parallel O \rightsquigarrow S' \parallel O'$ (cf. Definition 5.2, page 639); the proof is by induction on the derivation $\mathfrak{t}$. We prove that, for a system S''

$$(S \parallel O)\sigma \rightsquigarrow S'' \parallel O' \quad \wedge \quad S'\sigma \sqsubseteq_{\mathbf{m}} S''. \tag{30}$$

Notice that by (30), if there is a successful computation from $S \parallel O$, then there is one for $(S \parallel O)\sigma$ which proves the thesis.

Assume that $\mathfrak{t}$ is an instance of axiom `(os1)`, then (cf. Remark C.2) $\mathfrak{t}$ has the form $\Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![\pi\sharp Q.P \fatsemi R]\!] \parallel \overline{\pi}.O' \rightsquigarrow \Gamma \vdash \nu\tilde{x}\ \mathbb{C}[\![P \fatsemi R \mid Q]\!] \parallel O'$ therefore $(S \parallel O)\sigma = \Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}_\sigma[\![\pi\sharp Q.P \fatsemi R]\!] \parallel \overline{\pi}.O'$, by definition. Hence,

$$(S \parallel O)\sigma \quad \rightsquigarrow \quad \Gamma\sigma \vdash \nu\tilde{x}\ \mathbb{C}_\sigma[\![P \fatsemi R]\!] \parallel O'.$$

If $\mathfrak{t}$ is an instance of `(os2)` or `(os3)`, an argument similar to the previous one applies.

If $\mathfrak{t}$ is obtained with a derivation ending with an application of rule `(os4)`, the thesis follows by Proposition 7.3 or Corollary 7.2 – the former for cases (a), (c) and (d), and the latter for case (b) – if $\sigma \neq [{}^{\mathtt{rn}}/_{\mathtt{ns}}]^{\mathtt{i}}$ otherwise it follows by Proposition 7.4.

If $\mathfrak{t}$ is obtained with a derivation ending with an application of rule `(os5)`, then $O = O_1 + O_2$ and $S \parallel O_1 \rightsquigarrow S' \parallel O'_1$. By inductive hypothesis systems, there is a system S'' such that $S\sigma \parallel O_1 \rightsquigarrow S'' \parallel O'_1$ and $S'\sigma \sqsubseteq_{\mathbf{m}} S''$. By `(os5)`, $S\sigma \parallel O_1 + O_2 \rightsquigarrow S'' \parallel O'_1$ which yields the thesis.

Finally, the thesis trivially follows by the inductive hypothesis when $\mathfrak{t}$ is obtained with a derivation ending with an application of rule `(os6)`. □

References

Bocchi, L. (2004) Compositional nested long running transactions. In: Proceedings on the Fundamental Approaches to Software Engineering (FASE). *Springer Lecture Notes in Computer Science* **2984** 194–208.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Bocchi, L., Guanciale, R., Strollo, D. and Tuosto, E. (2010) Modelling transactional services in dynamically reconfiguring systems. In: Maglio, P., Weske, M., Yang, J. and Fantinato, M. (eds.) Proceedings of the 8th International Conference on Service Oriented Computing 2010. *Springer-Verlag Lecture Notes in Computer Science* **6470** 396–410.

Bocchi, L., Laneve, C. and Zavattaro, G. (2003) A calculus for long-running transactions. In: Najm, E., Nestmann, U. and Stevens, P. (eds.) Proceedings on Formal Methods for Open Object-Based Distributed Systems (FMOODS). *Springer Lecture Notes in Computer Science* **2884** 124–138.

Bocchi, L. and Lucchi, R. (2006) Atomic commit and negotiation in service oriented computing. In: Proceedings of *COORDINATION. Springer Lecture Notes in Computer Science* **4038** 16–27.

Bocchi, L. and Tuosto, E. (2010a) A Java inspired semantics for transactions in SOC. In: Wirsing, M., Hofmann, M. and Rauschmayer, A. (eds.) Proceedings of the 5th International Symposium on Trustworthy Global Computing. *Springer-Verlag Lecture Notes in Computer Science* **6084** 120–134.

Bocchi, L. and Tuosto, E. (2010b) Testing attribute-based transactions in soc. In: Hatcliff, J. and Zucca, E. (eds.) Formal Methods for Open Object-Based Distributed Systems and Formal Techniques for Networked and Distributed Systems (FMOODS/FORTE). *Springer-Verlag Lecture Notes in Computer Science* **6117** 87–94.

Bruni, R., Melgratti, H. and Montanari, U. (2004) Nested commits for mobile calculi: Extending Join. In: Lévy, J.-J., Mayr, E. and Mitchell, J. (eds.) *Proceedings of the 3rd IFIP International Conference on Theoretical Computer Science*. Kluwer 563–576.

Bruni, R., Melgratti, H.C. and Montanari, U. (2005) Theoretical foundations for compensations in flow composition languages. In: Symposium on Principles of Programming Languages (POPL). *Association for Computing Machinery* 209–220.

Buscemi, M.G. and Melgratti, H. C. (2007) Transactional service level agreement. In: Barthe, G. and Fournet, C. (eds.) Proceeding of Trustworthy Global Computing. *Springer Lecture Notes In Computer Science* **4912** 124–139.

Buscemi, M.G. and Montanari, U. (2007) Cc-pi: A constraint-based language for specifying service level agreements. In: Nicola, R. D. (ed.), Proceedings on the European Symposium on Programming (ESOP).*Springer Lecture Notes in Computer Science* **4421** 18–32.

Butler, M. and Ferreira, C. (2004) An operational semantics for StAC, a language for modelling long-running business transactions. In: De Nicola, R., Ferrari, G. and Meredith, G. (eds.) Proceeding of Coordination 2004. *Springer-Verlag Lecture Notes in Computer Science* **2949** 87–104.

Butler, M.J., Hoare, C.A.R. and Ferreira, C. (2004) A trace semantics for long-running transactions. In: Abdallah, A. E., Jones, C. B. and Sanders, J. W. (eds.) Proceedings on 25 Years of Communicating Sequential Processes. *Springer Lecture Notes in Computer Science* **3525** 133–150.

Dalal, S., Temel, S., Little, M., Potts, M. and Webber, J. (2003) Coordinating business transactions on the web. *IEEE Internet Computing* **7** (1) 30–39.

De Nicola, R. and Hennessy, M. C. B. (1984) Testing equivalences for processes. *Theoretical Computer Science* **34** (1–2) 83–133.

de Vries, E., Koutavas, V. and Hennessy, M. (2010a) Communicating transactions - (extended abstract). In: Gastin, P. and Laroussinie, F. (eds.) Proceedings on 21th Conference on International Concurrency Theory. *Springer Lecture Notes in Computer Science* **6269** 569–583.

de Vries, E., Koutavas, V. and Hennessy, M. (2010b) Liveness of communicating transactions (extended abstract). In: Proceedings of the APLAS, to appear.

EJB (2009) Enterprise Java Beans (EJB) technology. Sun Microsystems, `http://java.sun.com/products/ejb/`.

Guidi, C., Lanese, I., Montesi, F. and Zavattaro, G. (2009) Dynamic error handling in service oriented applications. *Fundamenta Informaticae* **95** (1) 73–102.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press

Igarashi, A., Pierce, B. and Wadler, P. (1999) Featherweight java - a minimal core calculus for java and gj. In: *ACM Transactions on Programming Languages and Systems* **23** (3) 396–450.

Lanese, I. (2010) Static vs dynamic sagas. In: Bliudze, S., Bruni, R., Grohmann, D. and Silva, A. (eds.) *Proceedings Third Interaction and Concurrency Experience (ICE 2010)*, EPTCS 38 51–65.

Lanese, I., Vaz, C. and Ferreira, C. (2010) On the expressive power of primitives for compensation handling. In: Gordon, A.D. (ed.) European Symposium on Programming. *Springer Lecture Notes in Computer Science* **6012** 366–386.

Laneve, C. and Zavattaro, G. (2005) Foundations of web transactions. In: Conference on Foundations of Software Science and Computation Structures (FoSSaCS). *Springer Lecture Notes in Computer Science* **3441** 282–298.

Mazzara, M. and Lanese, I. (2006) Towards a unifying theory for web services composition. In: WS-FM *Springer Lecture Notes in Computer Science* **4184** 257–272.

OASIS (2002) Business Transaction Protocol (BTP).

OASIS (2009) Web Services Transaction (WS-TX).

Panda, D., Rahman, R., and Lane, D. (2007) *EJB 3 in action*, Manning.

Vaz, C., Ferreira, C., and Ravara, A. (2008) Dynamic recovering of long running transactions. In: Proceedings on the Trustworthy Global Computing *Springer Lecture Notes in Computer Science* **5474** 201–215.

https://doi.org/10.1017/S0960129512000904 Published online by Cambridge University Press