Dynamic Recovering of Long Running Transactions

Cátia Vaz[1], Carla Ferreira[2], and António Ravara[3]*

[1] DEETC, ISEL, Polytechnic Institute of Lisbon, Portugal
[2] CITI and Dep. of Informatics, FCT, New University of Lisbon, Portugal
[3] SQIG, Instituto de Telecomunicações, and
Dep. of Mathematics, IST, Technical University of Lisbon, Portugal

Abstract. Most business applications rely on the notion of long running transaction as a fundamental building block. This paper presents a calculus for modelling long running transactions within the framework of the π-calculus, with support for compensation as a recovery mechanism. The underlying model of this calculus is the asynchronous polyadic π-calculus, with an extra layer for the definition of transaction scopes and dynamic installation of compensation processes. We add to the framework a type system which guarantees that transactions are unequivocally identified, ensuring that upon a failure the correct compensation process is invoked. Moreover, the operational semantics of the calculus ensures both installation and activation of the compensation of a transaction.

1 Introduction

Long Running Transactions (LRTs) are supported by most of the modern business applications. Due to the long running nature of business activities, traditional ACID transactions [1] are not suitable for them. Usually, LRTs are interactive and, consequently, cannot be check-pointed. Therefore, LRTs cannot be based on locking, as usual for traditional ACID transactions. Instead, they use *compensations*, which are activities programmed to recover partial executions of transactions.

Several business activities specifications have been proposed, such as active service WS-CAF [2] and WS-Coordination/Transaction [3], supporting the notion of LRTs. These transactions are a fundamental concept in building reliable distributed applications, namely when aggregating Web Services. Orchestration and Choreography languages, such as Microsoft XLANG [4] and its visual environment BizTalk, WS-BPEL [5], WS-CDL [6] and WSCI [6], allow the definition of complex services in terms of interactions among simpler services. However, each specification has a significative different interpretation of the notion of LRTs and compensable processes. Furthermore, in most specifications, this interpretation is defined informally by textual description.

* Partially supported by EC through project Sensoria and by FCT, through the Multiannual Funding Programme.

To proper model, and specially, to be able to reason and ensure properties about system specifications supporting LRTs, one needs mathematical tools. Process calculi is one suitable tool, providing not only a description language, but a rigorous semantics as well, allowing the proof of relevant properties.

There are several approaches that use process calculus towards the formalisation of LRTs. Such approaches either rely on, and try to model closely, existing standards and technologies [7–9], or are language and technology independent, focusing on the main concepts associated to LRTs [10–14]. However, in most of these works the compensation mechanism is static, and moreover, their settings do not provide an important property: the guarantee of installation and of activation of a compensation. In Section 7 we present, and compare in detail with, related work.

This paper defines a formal calculus to model the dynamic recovering of LRTs, which is language and technology independent. The calculus focus on the following main concepts: compensable activities, compensation scope, dynamic installation of compensations, nesting interruptible processes and failure handlers. The calculus is built on the framework of π-calculus with a compensable transaction mechanism. Thus, it is an extension of asynchronous polyadic π-calculus based on the fact that LRTs can be seen as interactive components, communicating asynchronously, within a distributed system.

Since our model is based on the interaction of a distributed system, we have found important that each step in such interaction would have associated a compensation process. One of the original ideas of our calculus is that, in each communication, the act of sending or receiving a message may have associated a compensation. Therefore the compensation of each transaction is *dynamically* built. In our model, the transaction either terminates successfully or fails. In case of failure, the stored compensations are automatically activated. The origin of the failure can be internal or external, and the compensation activation is transparent in the sense that there is no need to specify it explicitly in the calculus.

Another contribution is that the operational semantics gives priority to failure messages, which prevents failed transactions to continue executing. This, together with a type discipline to assert that transactions are unequivocally identified, ensures transaction soundness, *i.e.*, once a (internal or external) failure occurs, the compensation is activated and will eventually occur and terminate. Subject reduction ensures that this property is preserved by the operational semantics. Furthermore, the type discipline also ensures the absence of arity mismatches in input/output.

Proofs for the results presented herein can be found in a technical report [15].

2 Dynamic Compensation Calculus

This paper presents a calculus for modelling long running transactions, with a compensation mechanism to recover from failures. This mechanism allows to incrementally build the compensations of the transactions, within each inter-

$$
\begin{array}{lll}
P, Q \quad ::= & & \textit{Compensation Processes } (\mathcal{P}) \\
 & \mathbf{0} & \text{(Inaction)} \\
 & \mid \ \overline{a}\,\langle \tilde{v} \rangle & \text{(Output)} \\
 & \mid \ \sum_{i \in I} a_i(\tilde{x}_i).P_i & \text{(Input guarded choice)} \\
 & \mid \ (P \mid Q) & \text{(Parallel composition)} \\
 & \mid \ (\nu\, x)\, P & \text{(Restriction)} \\
 & & \\
\text{PP,QQ} \ ::= & & \textit{Compensable Processes } (\mathcal{CP}) \\
 & \mathbf{0} & \text{(Inaction)} \\
 & \mid \ \overline{a}\,\langle \tilde{v} \rangle\, \%P & \text{(Output)} \\
 & \mid \ \sum_{i \in I} a_i(\tilde{x}_i)\%P_i.PP_i & \text{(Input guarded choice)} \\
 & \mid \ (PP \mid QQ) & \text{(Parallel composition)} \\
 & \mid \ !a(\tilde{x})\%P.PP & \text{(Input guarded replication)} \\
 & \mid \ (\nu\, x)\, PP & \text{(Restriction)} \\
 & \mid \ a(\tilde{x})\,[PP] & \text{(Transaction scope)}
\end{array}
$$

Fig. 1. The syntax of processes.

action. To model this strategy, we distinguish between compensation processes (P,Q,...) and compensable processes (PP,QQ,...). The first are finite processes which can be associated to an act of sending or receiving a message; the others define processes that can be compensated.

A transaction $a(\tilde{x})\,[PP]$ encloses a compensable process PP within a transaction scope unequivocally identified by name a. A transaction either terminates successfully or fails. In case of failure, all stored compensations of the transaction are activated. The transaction identifier a is used for failure messages communication, which can be internal or external to the transaction.

The storing of the compensations is done within the execution of each step of the interaction. If a transition fails, its stored compensations cannot be interrupted by another failure.

The syntax of our language relies on a countable set of names, ranged over $a, b, c, x, y, z, \ldots$ and natural numbers, ranged over $i, j, k, \ldots$. Tuples are denoted by $\tilde{x}$. The grammar in Figure 1 defines the syntax of processes.

A compensation process can be: the inaction process $\mathbf{0}$; an output $\overline{a}\,\langle \tilde{v} \rangle$ that sends a tuple of names $\tilde{v}$ through the name a; an input guarded summation $\sum_{i \in I} a_i(\tilde{x}_i).P_i$ that after receiving a tuple of names $\tilde{v}$ through a_j, with $j \in I$, behaves like $P_j\{^{\tilde{v}}/_{\tilde{x}_j}\}$; a parallel composition of processes; or a restriction $(\nu\, x)\, P$ that behaves as P with the scope name x restricted to P. Note that compensation processes are finite.

In the case of a compensable process we have: the inaction process $\mathbf{0}$; an output $\overline{a}\,\langle \tilde{v} \rangle\, \%P$ that sends a tuple of names $\tilde{v}$ through the name a, storing the compensation process P; an input guarded choice $\sum_{i \in I} a_i(\tilde{x}_i)\%P_i.PP_i$ that after receiving a tuple of names $\tilde{v}$ through a_j, with $j \in I$, behaves like $PP_j\{^{\tilde{v}}/_{\tilde{x}_j}\}$ and stores the compensation process $P_j\{^{\tilde{v}}/_{\tilde{x}_j}\}$; a parallel composition of processes; an input guarded replication $!a(\tilde{x})\%P.PP$ that after receiving a tuple of names $\tilde{v}$ through a behaves like $PP\{^{\tilde{v}}/_{\tilde{x}}\}|!a(\tilde{v})\%P.PP$ and stores the compensation

E,D ::=		*Runtime Processes ($\mathcal{RP}$)*
	PP	(Compensable processes)
$\mid$	$\{PP\}$	(Stored compensation)
$\mid$	$E \mid D$	(Parallel composition)
$\mid$	$(\nu\, x)\, E$	(Restriction)
$\mid$	$\langle PP \rangle$	(Protected block)
$\mid$	$a(\tilde{x})\, [E]$	(Transaction scope)

Fig. 2. The runtime syntax of processes.

process $P\{^{\tilde{v}}/_{\tilde{x}}\}$; a restriction $(\nu\, x)\, PP$ that behaves as PP with the scope name x restricted to PP; or a transaction scope $a(\tilde{x})\, [PP]$ that behaves as PP until it receives a tuple of names $\tilde{x}$ through the name a, which will activate the stored compensations of PP.

The symbols $\mathbf{0}$, $.$, $\mid$ and ν are overloaded because they denote the same operators both in compensation and compensable processes. However, the actual meaning is made clear in each context.

For simplicity of the calculus, namely of the operational semantics, instead of storing a compensation process, we store a compensable one. This last is built from the compensation process by associating to each input/output the inaction process $\mathbf{0}$ as compensation.

The level of granularity of the dynamic installation of compensations in our model is very flexible. The system designer can choose from defining a compensation process for each execution step to defining a unique compensation for a rather complex process. The last case can be achieved by associating the inaction process as a compensation to all processes except for one, which will be the overall compensation of the transaction.

Processes at runtime achieve an extended syntax. Within the execution of some compensable processes, namely output, input guarded choice and input guarded replication, compensations are stored. Also, on the occurrence of a failure of a transaction, the respective stored compensations will be activated and placed within a protected block. This block ensures that compensations cannot be interrupted. The grammar in Figure 2 defines the syntax of runtime processes.

We use the standard notion of *bound names* for the compensation processes. We extend this notion to compensable and runtime processes such that in each $a(\tilde{x})\%P.PP$ the displayed occurrence of $\tilde{x}$ binds with scope PP and with scope P. Also, in $(\nu\, x)\, E$ and $a(\tilde{x})\, [E]$, x binds with scope E. Furthermore, we use the standard notions of *free names* of processes and of *α-equivalence*. We write $\mathsf{bn}(P)$ (respectively $\mathsf{fn}(P)$) for the set of names that are bound (respectively free) in a compensation process P. We use the same notation for compensable and runtime processes.

In the following we abbreviate the parallel composition of the processes P_i with $i \in I$, where I is a finite set of indexes, with $\Pi_{i \in I} P_i$. When the meaning is clearly in the context, we will abbreviate $a(\tilde{x})\%0$ and $\overline{a}\, \langle \tilde{v} \rangle\, \%0$ with $a(\tilde{x})$ and $\overline{a}\, \langle \tilde{v} \rangle$, respectively.

3 Example

In this section we describe an ordering system. We can think of it as a web shop that accepts orders from clients. Whenever a client submits an order, the system must take care of payments and order packing. We model the system as depicted in Figure 3, which for simplicity only considers one client and one order.

$$\mathsf{\textbf{OrderTransaction}} \overset{\text{def}}{=} \mathsf{\textbf{Client}} \mid \mathsf{\textbf{Shop}} \mid \mathsf{\textbf{Bank}} \mid \mathsf{\textbf{Warehouse}}$$

$$\mathsf{\textbf{Client}} \overset{\text{def}}{=} \overline{client}\langle ord\rangle \mid (ack.\overline{s}\langle clientCancel\rangle + ack.\overline{b}\langle clientCancel\rangle + ack.done)$$

$$\mathsf{\textbf{Shop}} \overset{\text{def}}{=} s(x)\big[client(ord).(\overline{ack} \mid \mathsf{\textbf{Charge}} \mid \mathsf{\textbf{Pack}} \mid bankOk.packOk.\overline{done})\big]$$

$$\begin{aligned}
&\mathsf{\textbf{Charge}} \overset{\text{def}}{=} c\big[\\
&\quad(\overline{bank}\langle ord\rangle\%(\nu y)(\overline{y} \mid y.\overline{b}\langle shopCancel\rangle + y.\overline{s}\langle shopCancel\rangle)) \mid\\
&\quad(valid\%\overline{refund}\langle ord\rangle).\overline{bankOk} + invalid.\overline{s}\langle shopCancel\rangle\\
&\big]
\end{aligned}$$

$$\begin{aligned}
&\mathsf{\textbf{Pack}} \overset{\text{def}}{=} p\big[\\
&\quad(\overline{pack}\langle ord\rangle\%\overline{w}\langle shopCancel\rangle) \mid\\
&\quad(ok\%\overline{unpack}\langle ord\rangle).\overline{packOk} + notOk.\overline{s}\langle shopCancel\rangle\\
&\big]
\end{aligned}$$

$$\begin{aligned}
&\mathsf{\textbf{Bank}} \overset{\text{def}}{=} b(x)\big[\\
&\quad(bank(ord)\%(\overline{x} \mid clientCancel.\overline{c} + shopCancel))\ .\\
&\quad(\nu y)(\overline{y} \mid y.(\overline{valid}\%refund(ord)) + y.\overline{invalid})\\
&\big]
\end{aligned}$$

$$\mathsf{\textbf{Warehouse}} \overset{\text{def}}{=} w(x)\big[pack(ord).(\nu y)(\overline{y} \mid y.(\overline{ok}\%unpack(ord)) + y.\overline{notOk})\big]$$

Fig. 3. Ordering system example.

In this scenario, the client submits an order to the shop and waits for order confirmation. The shop receives the message and starts two transactions, charging the client and packing the order. The payment is done by the bank, therefore the shop sends a message to the bank and it starts a new transaction. Similarly, the shop sends a message to the warehouse and a new packing transaction starts. Notice that the client may cancel either the shop transaction or the bank transaction. In both cases, the system stops and compensation transactions are executed. If charging and packing are successfully accomplished, the shop sends a message to the client and terminates the transaction.

The compensations are incrementally built. For example, when the bank receives the message of the shop, it installs the compensation $(\overline{x} \mid \mathit{clientCancel}.\overline{c} + \mathit{shopCancel})$ and, when it validates the payment, it installs the compensation $\mathit{refund}(\mathit{ord})$. The first compensation checks the cancellation message acting accordingly and the second one waits for a refund. It is interesting to note that if the bank fails before charging validation, only the first compensation is executed. This is an important feature of our dynamic installation mechanism.

As noted before, the client can cancel both shop transaction and bank transaction. At this level it becomes handy to be able to distinguish different types of failure. If the client cancels the bank transaction, we must ensure that shop knows about it. Therefore, the compensation $(\overline{x} \mid \mathit{clientCancel}.\overline{c} + \mathit{shopCancel})$ checks who have interrupted the transaction and, if it has been the client, a failure message is sent through c. Thus, it is ensured that the client does not get the goods for free.

In this example we have also nested transactions. The shop transaction includes two inner transactions, the charging transaction and the packing transaction. If the shop transaction fails, then both inner transactions will also fail and, most important, we do not need to explicitly model it. Nesting also allows us to separate different inner transactions and restrict the action of external agents. In the example, the bank can only interrupt the inner transaction identified by c and, in spite of we choose to not do that, the shop could try another payment method without failing.

Finally, we refer the importance of keeping installed compensations after the end of a transaction. Suppose that the bank transaction ends successfully, but that the warehouse fails the packing. We must be able to compensate the charging transaction, i.e., we must refund the client. Later we will discuss how our calculus supports this behaviour.

4 Operational Semantics

The semantics for the language is introduced in this section, following the usual approach of Milner *et al.* [16] . We define an operational semantics by means of a reduction relation on processes, making use of a structural congruence relation.

Definition 1. *Structural congruence $\equiv$ is defined as the smallest congruence relation on compensable and executable processes satisfying α-conversion law, abelian monoid laws for parallel and the following laws:*

1. *Scope extension laws:*

$$\begin{array}{rcll}
(\nu\, x)\, \mathbf{0} & \equiv & \mathbf{0} & \\
a(\tilde{x})[\,(\nu\, y)\, E\,] & \equiv & (\nu\, y)\, (a(\tilde{x})\, [E]) & \textit{if} \quad y \notin \tilde{x} \\
(\nu\, z)\, (\nu\, w)\, E & \equiv & (\nu\, w)\, (\nu\, z)\, E & \\
E \mid (\nu\, z)\, D & \equiv & (\nu\, z)\, (E \mid D) & \textit{if} \quad z \notin \mathsf{fn}(E) \\
\langle (\nu\, x)\, PP \rangle & \equiv & (\nu\, x)\, \langle PP \rangle &
\end{array}$$

2. Transaction scope and protected block laws:

$$a(\tilde{x})\,[E \mid \langle PP\rangle] \equiv a(\tilde{x})\,[E] \mid \langle PP\rangle \qquad \langle PP \mid QQ\rangle \equiv \langle PP\rangle \mid \langle QQ\rangle$$

3. Termination laws:

$$\langle \mathbf{0}\rangle \equiv \mathbf{0} \qquad \mathbf{0}\%\mathbf{0} \equiv \mathbf{0} \qquad \{\mathbf{0}\} \equiv \mathbf{0}$$

The scope and termination laws are standard. The transaction scope law moves protected blocks outside from transactions, since they cannot be interrupted.

The dynamic behaviour of processes is defined by a reduction relation. To define this relation, we must take into account some aspects related to the transaction scope behaviour. For instance, when we send a message through the transaction identifier, the transaction should fail and all its stored compensations should be activated. Therefore, we have to extract the stored compensations and place them in a protected block. To extract stored compensations of a transaction scope, we use the function extr which is defined as follows.

Definition 2. *Function* $\mathsf{extr} : \mathcal{RP} \longmapsto \mathcal{RP}$ *for extracting stored compensations, is inductively defined as:*

$$\begin{aligned}
\mathsf{extr}(PP) &= \mathbf{0}\\
\mathsf{extr}(\{PP\}) &= PP\\
\mathsf{extr}(a(\tilde{x})\,[E]) &= \mathsf{extr}(E)\\
\mathsf{extr}(\langle PP\rangle) &= \langle PP\rangle\\
\mathsf{extr}(E \mid D) &= \mathsf{extr}(E) \mid \mathsf{extr}(D)\\
\mathsf{extr}((\nu\, x)\, E) &= (\nu\, x)\ \mathsf{extr}(E)
\end{aligned}$$

With the above definition, we can see that a given runtime process E, $\mathsf{extr}(E)$ is structurally congruent to a process of the form $(\nu\,\tilde{y})\,(\Pi_{i\in I} QQ_i \mid \Pi_{j\in J}\langle RR_j\rangle)$. In fact, this result can be easily proven by structural induction on E.

In our semantics, we also want to give priority to transaction identifiers, to prevent the execution of already failed transactions. Therefore, it is necessary to recognize the *set of transaction identifiers* that occur in a runtime process E. Namely, in each $a(\tilde{x})[D]$, the displayed occurrence of a is a *transaction identifier*. Given a runtime process E, the set $\mathsf{ti}(E)$ is defined as the *set of transaction identifiers* that occur in E. Notice that the set of transaction identifiers of a compensable process PP is similarly defined and that there are no transaction identifiers on compensation processes.

In order to give priority to transactions failures, we must define another set of names. Given a runtime process E, we define $\mathsf{rdy}(E)$ as the *set of names in E that are ready to communicate*: if an output $\overline{a}\,\langle\tilde{v}\rangle\,\%Q$ and an input $a(\tilde{x})\%P$ are subterms of a runtime process E with no prefixing terms, *i.e.*, there is no QQ such that $QQ.\overline{a}\,\langle\tilde{v}\rangle\,\%Q$ or $QQ.a(\tilde{x})\%P$ are subterms of E, the name a is *ready to communicate*; if an output $\overline{a}\,\langle\tilde{v}\rangle\,\%Q$ and a transaction scope $a(\tilde{x})\,[D]$ are subterms of a runtime process E with no prefixing terms, the name a is *ready to communicate*.

(R-recover-out)

$$\frac{\mathsf{extr}(E) \equiv (\nu\,\tilde{y})\,(\prod_{i=0}^{n} QQ_i \;\mid\; \prod_{j=0}^{m} \langle RR_j \rangle)}{\overline{a}\,\langle \tilde{z} \rangle\,\%P \mid a(\tilde{x})\,[E] \rightarrow \langle (\nu\,\tilde{y})\,(\prod_{i=0}^{n} QQ_i\{^{\tilde{z}}/_{\tilde{x}}\} \mid \prod_{j=0}^{m} \langle RR_j \rangle)\rangle \mid \{\varphi(P)\}}$$

(R-recover-in)

$$\frac{\mathsf{extr}(E) \equiv (\nu\,\tilde{y})\,(\prod_{i=0}^{n} QQ_i \;\mid\; \prod_{j=0}^{m} \langle RR_j \rangle)}{a(\tilde{x})\,[E \mid \overline{a}\,\langle \tilde{z} \rangle\,\%P] \rightarrow \langle (\nu\,\tilde{y})\,(\prod_{i=0}^{n} QQ_i\{^{\tilde{z}}/_{\tilde{x}}\} \mid \prod_{j=0}^{m} \langle RR_j \rangle) \mid \varphi(P)\rangle}$$

(R-recover-scope)

$$\frac{a \in \mathsf{rdy}(F) \quad a(\tilde{x})\,[E \mid F] \rightarrow E'}{a(\tilde{x})\,[E \mid b(\tilde{z})\,[F]] \rightarrow E'}$$

(R-res)

$$\frac{E \rightarrow E'}{(\nu\,x)\,E \rightarrow (\nu\,x)\,E'}$$

(R-struct)

$$\frac{E' \equiv E \quad E \rightarrow D \quad D \equiv D'}{E' \rightarrow D'}$$

(R-scope)

$$\frac{E \rightarrow E' \quad a \notin \mathsf{rdy}(E)}{a(\tilde{x})[\,E\,] \rightarrow a(\tilde{x})[\,E'\,]}$$

(R-scope-com)

$$\frac{E \mid D \rightarrow E' \mid D' \quad a \notin \mathsf{rdy}(E \mid D)}{a(\tilde{x})[\,E\,] \mid D \rightarrow a(\tilde{x})[\,E'\,] \mid D'}$$

(R-block)

$$\frac{PP \rightarrow PP'}{\langle PP \rangle \rightarrow \big\langle PP' \big\rangle}$$

(R-block-com)

$$\frac{PP \mid E \rightarrow PP' \mid E'}{\langle PP \rangle \mid E \rightarrow \big\langle PP' \big\rangle \mid E'}$$

(R-par-t)

$$\frac{E \rightarrow E' \quad \mathsf{ti}(D) \cap \mathsf{rdy}(D) = \emptyset}{E \mid D \rightarrow E' \mid D}$$

(R-par-c)

$$\frac{E \;\rightarrow\; E' \quad \mathsf{ti}(D) \cap \mathsf{rdy}(D) \neq \emptyset \quad \mathsf{ti}(E) \cap \mathsf{rdy}(E) \neq \emptyset}{E \mid D \;\rightarrow\; E' \mid D}$$

(R-com)

$$\frac{j \in I}{\overline{a_j}\langle \tilde{v} \rangle\%Q \mid \sum_{i \in I} a_i(\tilde{x}_i)\%P_i.PP_i \;\rightarrow\; \{\varphi(Q)\} \mid \{\varphi(P_j\{^{\tilde{v}}/_{\tilde{x}_j}\})\} \mid PP_j\{^{\tilde{v}}/_{\tilde{x}_j}\}} \quad \forall_{i \in I}\, |\tilde{v}| = |\tilde{x_i}|$$

(R-rep)

$$\frac{}{\overline{a_j}\langle \tilde{v} \rangle\%Q|!a(\tilde{x})\%P.PP \rightarrow \{\varphi(Q)\} \mid \{\varphi(P\{^{\tilde{v}}/_{\tilde{x}}\})\} \mid PP\{^{\tilde{v}}/_{\tilde{x}}\}|!a(\tilde{x})\%P.PP} \quad |\tilde{v}| = |\tilde{x}|$$

Fig. 4. Reduction rules.

Definition 3. *The reduction relation $\rightarrow$ is the least relation satisfying the rules of Figure 4.*

The rules in Figure 4 describe the execution of the system. Rules R-res and R-struct are standard in process calculus. In rules R-recover-in, R-recover-out, R-com and R-rep we use the function $\varphi : \mathcal{P} \longmapsto \mathcal{CP}$ to map compensation processes to compensable ones, associating to each input/output an inaction process as compensation. Rules R-recover-in and R-recover-out model transaction failures: when a transaction receives a failure message, compensations will be extracted and activated, placing them into a protected block. The block ensures that no other failure can interrupt these compensations. R-Recover-in is used when the failure message is internal to the transaction and R-recover-out otherwise. There is also another rule that model transaction failures, namely R-recover-scope. In this rule, the failure message is internal to the transaction, but it is nested within another transaction block. Notice that an interaction can occur inside the same context or in different contexts. Therefore, we use rules R-block and R-scope for standard execution of a process inside a transaction scope and a protected block, respectively. Rules R-block-com and R-scope-com are needed for an interaction occurring between different contexts.

Instead of the standard rule for parallel, we have defined two rules, R-par-c and R-par-t. With both rules, we have defined a semantics that gives priority to failure messages over all other messages. Rules R-com and R-rep allow communication, while storing the associated compensations.

5 Uniqueness of Transaction Identifiers

This section presents a type system for our language. It was developed to guarantee that transactions are unequivocally identified. Therefore, it can be ensured that upon a failure, the correct compensation is invoked. Thus, there are two important aspects to consider. Firstly, transaction identifiers must be unique, *i.e.*, it cannot exist two transaction scopes with the same identifier. Consequently, we must ensure that in the case of the input guarded replication $!a(\tilde{x})\%P.PP$, the transaction identifiers cannot occur free. This leads to the definition of the *set of free transaction identifiers* of a process, which results from the intersection of the set of bound names with the set of transaction identifiers of the process. We denote by $\mathsf{fti}(E)$ the *set of free transaction identifiers* of a runtime process E. The set of free transaction identifiers of a compensable process is similarly defined.

In order to ensure that transactions are unequivocally identified is also necessary to distinguish their identifiers from other names in our type system, namely input names. The *set of input names* that occur in a compensation process P, denoted by $\mathsf{in}(P)$, is composed by each name a that occurs as $a(\tilde{x})$, subterm of P. Furthermore, in each occurrence of $a(\tilde{x})\%P$ in a runtime process E, the name a and the input names of P are included in the *input names* of E. As well, in each occurrence of $b\,\langle\tilde{v}\rangle\,\%P$ in E the input names of P are included in the *input names* of E. Given a runtime process E, the set $\mathsf{in}(E)$ is defined as the *set of input names* that occur in E. The set of input names for a compensable process is similarly defined.

In our calculus, the names $\tilde{x}$ in a transaction scope $a(\tilde{x})\,[E]$ are useful to activate different compensations in case of the transaction failure, depending on the failure message. Therefore, the names $\tilde{x}$ should only occur in the compensation processes of E. Even more, the names $\tilde{x}$ should not occur free in the compensations of inner transactions of E. The design choice of an inner transaction should

$$\begin{array}{rcll}
 & & & \textit{Type for Names} \\
V & ::= & X & \text{(Variable Type)} \\
 & \mid & ch(V_1, ..., V_n),\ n \geq 0 & \text{(Channel Types)} \\
 & & & \\
 & & & \textit{Types} \\
S & ::= & V & \text{(Type for Names)} \\
 & \mid & tc(V_1, ..., V_n),\ n \geq 0 & \text{(Types for Transaction Channels)} \\
 & & & \\
 & & & \textit{Type Environments} \\
\Gamma & ::= & \Gamma, x : S & \\
 & \mid & \emptyset &
\end{array}$$

Fig. 5. The syntax of types.

not have access to failure messages from the outer transaction ensures the scope separation among transactions. To represent this feature it is necessary to identify the *set of execution names* that occur in a runtime process E. Namely, in each occurrence of $a(\tilde{x})\%P$ in E, the name a is included in the *execution names* of E. As well, in each occurrence of $a(\tilde{x})\,[D]$, the name a and all names of D are included in the *execution names* of E. Also, for each occurrence $\overline{a}\,\langle\tilde{x}\rangle\,\%P$, the names a and $\tilde{x}$ are included in the *execution names* of E. The *set of free execution names* of a runtime process E, denoted by $\mathsf{fen}(E)$, results from the intersection of the set of the bound names with the set of execution names of E.

Definition 4. *The grammar in Figure 5 defines the syntax of types.*

We assume a countable set $X, Y, \ldots$ of type variables and a countable set $x, y, \ldots$ of labels. We write $\tilde{V}$ for a tuple $V_1, \ldots, V_n$ of types and $\tilde{v} : \tilde{V}$ for a sequence $v_1 : V_1, \ldots, v_n : V_n$ of labelled types. When we do not want to distinguish type names, we use the letter S. The type system can now be defined.

Definition 5. *The rules in Figure 6 inductively define the type system.*

In the specification of the type system we have assumed a countable set of names $a, b, x, y, z, \ldots$ which is disjoint from the set of labels. The type system rules given in Figure 6 define judgements of the form $\Gamma \vdash \tilde{v} : \tilde{V}$, $\Gamma \vdash a : ch(\tilde{V})$, $\Gamma \vdash a : tc(\tilde{V})$, $\Gamma \vdash P$ or $\Gamma \vdash E$, where Γ is an environment, P a compensation process and E a runtime process. The judgements $\Gamma \vdash a : ch(\tilde{V})$ and $\Gamma \vdash a : tc(\tilde{V})$ state that variable a is of type channel and transaction channel, respectively. Process type judgements such as $\Gamma \vdash P$ and $\Gamma \vdash E$ state that process P and E respect the type assumptions in Γ. A process is either correctly typed or not. This type system is consistent with the operational semantics and ensures that transactions are unequivocally identified. These results are stated bellow. First we need to define the predicate $\mathsf{unq}(E)$ in order to state the uniqueness property of the transaction identifiers.

(T-NAME) $\Gamma, x : X \vdash x : X$ (TP-NIL) $\Gamma \vdash 0$

(TP-PAR) $$\frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \,|\, Q}$$

(TP-OUT-C) $$\frac{\Gamma \vdash a : ch(\tilde{V}) \quad \Gamma \vdash \tilde{v} : \tilde{V}}{\Gamma \vdash \overline{a}\,\langle \tilde{v} \rangle}$$

(TP-OUT-T) $$\frac{\Gamma \vdash a : tc(\tilde{V}) \quad \Gamma \vdash \tilde{v} : \tilde{V}}{\Gamma \vdash \overline{a}\,\langle \tilde{v} \rangle}$$

(TP-INP) $$\frac{\forall_{i \in I}(\ \Gamma \vdash a_i : ch(\tilde{V}_i) \quad \Gamma, \tilde{x_i} : \tilde{V}_i \vdash P_i\ )}{\Gamma \vdash \sum_{i \in I} a_i(\tilde{x}_i).P_i}$$

(TP-RES) $$\frac{\Gamma, x : V \vdash P}{\Gamma \vdash (\nu\, x)\, P}$$

(TE-NILL) $\Gamma \vdash 0$

(TE-OUT-C) $$\frac{\Gamma \vdash a : ch(\tilde{V}) \quad \Gamma \vdash \tilde{v} : \tilde{V} \quad \Gamma \vdash Q}{\Gamma \vdash \overline{a}\,\langle \tilde{v} \rangle\, \%Q}$$

(TE-OUT-T) $$\frac{\Gamma \vdash a : tc(\tilde{V}) \quad \Gamma \vdash \tilde{v} : \tilde{V} \quad \Gamma \vdash Q}{\Gamma \vdash \overline{a}\,\langle \tilde{v} \rangle\, \%Q}$$

(TE-INP) $$\frac{\forall_{i \in I}(\ \Gamma \vdash a_i : ch_i(\tilde{V}_i) \quad \Gamma, \tilde{x_i} : \tilde{V}_i \vdash PP_i \quad \Gamma, \tilde{x_i} : \tilde{V}_i \vdash Q_i\ )}{\Gamma \vdash \sum_{i \in I} a_i(\tilde{x}_i)\%Q_i.PP_i}$$

(TE-REP) $$\frac{\Gamma \vdash a : ch(\tilde{V}) \quad \Gamma, \tilde{x} : \tilde{V} \vdash PP \quad \Gamma, \tilde{x} : \tilde{V} \vdash Q \quad \mathsf{fti}(PP) = \emptyset}{\Gamma \vdash !a(\tilde{x}\%Q).PP}$$

(TE-RES) $$\frac{\Gamma, x : S \vdash E}{\Gamma \vdash (\nu\, x)\, E}$$

(TE-STORED) $$\frac{\Gamma \vdash PP}{\Gamma \vdash \{PP\}}$$

(TE-BLOCK) $$\frac{\Gamma \vdash PP}{\Gamma \vdash \langle PP \rangle}$$

(TE-PAR) $$\frac{\Gamma \vdash E \quad \Gamma \vdash D \quad \mathsf{fti}(D) \cap \mathsf{fti}(E) = \emptyset}{\Gamma \vdash E \,|\, D}$$

(TE-SCOPE) $$\frac{\Gamma, \tilde{x} : \tilde{V} \vdash E \quad \Gamma \vdash a : tc(\tilde{V}) \quad a \notin \mathsf{fti}(E) \quad \tilde{x} \notin \mathsf{fen}(E) \quad \forall_{b \in \mathsf{ti}(E)}\ |b| = 0}{\Gamma \vdash\ a(\tilde{x})\,[E]}$$

Fig. 6. Type system.

Definition 6. *The predicate* $\mathsf{unq}(E)$ *for runtime processes verifies if the transaction identifiers of a process are unique and is defined as:*

$$\begin{array}{rl}
\mathsf{unq}(0) & \\
\mathsf{unq}(\overline{a}\,\langle \tilde{v} \rangle\,\%Q) & \\
\mathsf{unq}(\sum_{i \in I} a_i(\tilde{x}_i)\%Q_i.PP_i) & \textit{if}\ \ \mathsf{unq}(PP_i),\ \textit{for all}\ i \in I \\
\mathsf{unq}(!a(\tilde{v})\%Q.PP) & \textit{if}\ \ \mathsf{fti}(PP) = \emptyset \\
\mathsf{unq}(\langle PP \rangle) & \\
\mathsf{unq}(\{PP\}) & \\
\mathsf{unq}(E \,|\, D) & \textit{if}\ \ \mathsf{unq}(E)\ \textit{and}\ \mathsf{unq}(D)\ \textit{and}\ \mathsf{fti}(E) \cap \mathsf{fti}(D) = \emptyset \\
\mathsf{unq}((\nu\, x)\, E) & \textit{if}\ \ \mathsf{unq}(E) \\
\mathsf{unq}(a(\tilde{x})[E]) & \textit{if}\ \ \mathsf{unq}(E)\ \textit{and}\ a \notin \mathsf{fti}(E)
\end{array}$$

Theorem 1. (Subject Reduction) *Let E be a process such that $\Gamma \vdash E$ and $E \rightarrow E'$. Then $\Gamma \vdash E'$.*

The following theorem asserts that transaction identifiers are unique and that they cannot be confused with input names. Thus, since we prioritize failure communications, we can ensure activation and execution of compensations as we discuss in the next section.

Theorem 2. (Soundness) *Let E be a runtime process such that $\Gamma \vdash E$. Then $\mathsf{in}(E) \cap \mathsf{ti}(E) = \emptyset$ and $\mathsf{unq}(E)$ holds.*

Notice that our type system assures that each name respects arity; that is, if the name a has arity n then each occurrence of $\overline{a}\,\langle x_1, \ldots, x_k\rangle$ and $a(x_1, \ldots, x_k)$ is well-formed only if $k = n$. A process is said to be well typed if it respects arity, *i.e.*, not fails. Therefore, our calculus is equipped with a safe type system, *i.e.*, well typed processes cannot fail.

Theorem 3. (Type Safety) *Let E be an execution process such that $\Gamma \vdash E$ and $E \rightarrow E'$. Then every name in E' respects arity.*

6 Properties of the Recovery Mechanism

In the reduction semantics of our calculus, some conditions were introduced in order to ensure both installation and activation of process compensations. In this section we formally state that compensations are installed within process execution and that they are automatically activated whenever a failure occurs.

First we introduce some terminology, namely the notion of execution context. Note that part of a process being executed can occur in different contexts.

Definition 7. *The grammar in Figure 7 inductively defines* execution contexts*, denoted by $C[\![\bullet]\!]$, and* double execution contexts*, denoted by $D[\![\bullet,\bullet]\!]$.*

$$\begin{array}{rcl} C[\![\bullet]\!] & ::= & \bullet \;\mid\; (\nu\, x)\, C[\![\bullet]\!] \;\mid\; C[\![\bullet]\!] \,|\, E \;\mid\; E \,|\, C[\![\bullet]\!] \;\mid\; a(\tilde{x})\,[C[\![\bullet]\!]] \;\mid\; \langle C[\![\bullet]\!]\rangle \\ D[\![\bullet,\bullet]\!] & ::= & C[\![\bullet]\!] \,|\, C[\![\bullet]\!] \end{array}$$

Fig. 7. Execution contexts and double execution contexts.

In our semantics, we are not only interested in installing compensations, but we also want to give priority to transaction identifiers. By prioritizing communications through transaction identifiers we are preventing the execution of transactions that have already failed. A transaction may fail in two different ways: the failure can be raised by internal or external messages. In the following propositions, we ensure that in both cases communication through transaction identifiers have priority to all other communications. Therefore, in case of failure, stored compensations are activated.

Proposition 1. *Let $E = a(\tilde{x})\,[F \mid C[\![\overline{a}\,\langle\tilde{y}\rangle\,\%Q]\!]$ be a typable process, for a context C. Suppose that $\mathsf{ti}(E)\cap\mathsf{rdy}(E) = \{a\}$. If $E \to E'$, then $E' = \langle F' \,|\, C'[\![\varphi(Q)]\!]\rangle$, for a context C', with $F' = (\nu\,\tilde{y})\,(\Pi_{i\in I}QQ_i\{\tilde{y}/\tilde{x}\} \mid \Pi_{j\in J}\langle RR_j\rangle)$ and $\mathsf{extr}(F) \equiv (\nu\,\tilde{y})\,(\Pi_{i\in I}QQ_i \mid \Pi_{j\in J}\langle RR_j\rangle)$.*

Proposition 2. *Let $E = D[\![a(\tilde{x})\,[F]\,,\overline{a}\,\langle\tilde{y}\rangle\,\%Q]\!]$ be a typable process for a context D. Suppose that $\mathsf{ti}(E)\cap\mathsf{rdy}(E) = \{a\}$. If $E \to E'$ then $E' = D'[\![\langle F'\rangle, \{\varphi(Q)\}]\!]$ for a context D', with $F' = (\nu\,\tilde{y})\,(\Pi_{i\in I}QQ_i\{\tilde{y}/\tilde{x}\} \mid \Pi_{j\in J}\langle RR_j\rangle)$ and $\mathsf{extr}(F) \equiv (\nu\,\tilde{y})\,(\Pi_{i\in I}QQ_i \mid \Pi_{j\in J}\langle RR_j\rangle)$.*

Whenever an interaction occurs, we must ensure that the associated compensation processes are installed. Since an input can occur in two situations, input guarded choice and input guarded replication, in the following propositions we state that in both cases the compensations are installed.

Proposition 3. *Let $D[\![\bullet,\bullet]\!]$ be a context where the name b does not occur, $E = D[\![\sum_{i\in I} a_i(\tilde{x_i})\%P_i.PP_i, \overline{a}_j\,\langle\tilde{y}\rangle\,\%Q]\!]$ be a typable process such that $a_j = b$, $\mathsf{rdy}(E) = \{a_j\}$ and $\mathsf{ti}(E)\cap\mathsf{rdy}(E) = \emptyset$. If $E \to E'$, then $E' = D'[\![PP_j\{\tilde{y}/\tilde{x}_j\} \mid \{\phi(P_j\{\tilde{y}/\tilde{x}_j\})\}, \{\phi(Q)\}]\!]$, for a context D'.*

Proposition 4. *Let $D[\![\bullet,\bullet]\!]$ be a context where the name b does not occur, $E = D[\![!b(\tilde{x})\%P.PP, \overline{b}\,\langle\tilde{y}\rangle\,\%Q]\!]$ be a typable process, $\mathsf{rdy}(E) = \{b\}$ and $\mathsf{ti}(E)\cap\mathsf{rdy}(E) = \emptyset$. If $E \to E'$, then $E' = D'[\![!b(\tilde{x})\%P.PP \mid PP\{\tilde{y}/\tilde{x}\} \mid \{\phi(P\{\tilde{y}/\tilde{x}\})\}, \{\phi(Q)\}]\!]$, for a context D'.*

7 Related Work

There are other approaches that use process calculus toward the formalization of LRTs and their compensations mechanisms. In this section we briefly compare our work with them.

Bocchi *et al.* introduced πt-calculus [7], which is inspired in BizTalk, and consists of an extension of asynchronous polyadic π-calculus [17] with the notion of transactions. However, the compensation of each transaction is statically defined, *i.e.*, the compensations are not incrementally built. In contrast, we have designed a dynamic recover mechanism.

The cJoin calculus [12] is an extension of Join calculus [18] with primitives for representing transactions. As in πt-calculus, the compensation mechanism of this calculus is statically defined. In contrast to our calculus, completed transactions cannot be compensated, *i.e.*, after a transaction completes, compensations are discarded. Therefore, in cJoin, only running transactions can be compensated whenever interrupted.

Butler and Ferreira [8] propose the StAC language, which is inspired in BP-Beans. The language includes the notion of compensation pair, similar to the sagas concept defined by Gargia-Molina and Salem [19]. In StAC, a LRT is seen as a composition of one or more sub-transactions, where each of them has an associated compensation. In contrast to our calculus, StAC is flow composition based and includes explicit operators for running or discarding installed compensations. As well, compensating CSP [10], denoted by cCSP, and Sagas calculi [11] are also composition flow based, namely they adopt a centralized coordination mechanism. They have similar operators but different compensation policies. However these three approaches are conceptually different from ours, as they are flow based and do not provide mobility.

Laneve and Zavattaro define a calculus named webπ [13] which is an extension of asynchronous polyadic π-calculus with a timed transaction construct. An untimed version of webπ, known as webπ_∞ was proposed by Mazzara and Lanese [14]. Although our calculus shares some syntax similarities with both calculus, we have followed different principles. Namely, in both calculus the nested transactions are flattened. This is a substantial difference with respect to our calculus, since it implies that the internal transaction cancelling must be explicitly programmed within the specification. Another difference is that in these calculus completed transactions cannot be compensated. As in πt-calculus the compensation mechanism is statically defined. Furthermore, they assume that transactions are unequivocally identified, whereas in our approach we have a type system to ensure this feature.

Guidi *et al.* [9] propose an extension of SOCK [20], which is inspired in WSDL and BPEL. This calculus includes explicit primitives for dynamic handler installation, such as fault and compensation handlers and automatic failure notification. They assert correctness properties for their calculus, namely the expected behaviour of a scope, the correct termination upon a failure, the correct behaviour of communications and guarantee of fault activation. Our approach is different in the sense that both installation and activation of compensations are transparent to the user, *i.e.*, they occur within interactions.Thus, making the syntax clear and simpler. Similar to $\mathtt{web}\pi_\infty$, they only assume that transactions are unequivocally identified, lacking support to formally ensuring this feature. Another difference is that in our calculus we use a type discipline to ensure soundness and the installation and activation of transaction compensations.

8 Conclusion

In this paper we have proposed a calculus for reasoning about long running transactions, language and technology independent. We have built our calculus on the framework of asynchronous polyadic π-calculus. One of our main contributions is a dynamic recovery mechanism based on compensations, supporting both incremental building of compensations and failure handling in a nested way. Another contribution is a type discipline that ensures both calculus soundness and safety. Finally, we have defined a compensation semantics that ensures both installation and activation of transaction compensations.

The expressiveness of our calculus was demonstrated with a case study. It was shown that is possible to model deeply connected transactions in a comprehensive way. Notice also that in the case study the overall effect of the execution of compensations is equivalent to not start the transactions. However, in a more complex scenario, it is hard to assert such behaviour. Thus, one of our future research directions is to study compensation soundness given a notion of transaction equivalence.

References

1. Gray, J.: The transaction concept: Virtues and limitations (invited paper). In: VLDB, IEEE Computer Society (1981) 144–154
2. OASIS: Web services composite application framework (ws-caf). (2005)
3. Microsoft, IBM, BEA: WS-coordination/WS-transaction specification. (2005)
4. Thatte, S.: XLANG: Web services for business process design. Technical report, Microsoft Corporation (2001)
5. OASIS: Web services business process execution language version 2.0. (April 2007)
6. Kavantzas, N., Olsson, G., Michkinsky, J., Chapman, M.: Web services choreography description language. Technical report, Oracle Corporation (2003)
7. Bocchi, L., Laneve, C., Zavattaro, G.: A calculus for long-running transactions. In Najm, E., Nestmann, U., Stevens, P., eds.: FMOODS. Volume 2884 of LNCS., Springer (2003) 124–138
8. Butler, M.J., Ferreira, C.: An operational semantics for StAC, a language for modelling long-running business transactions. In Nicola, R.D., Ferrari, G.L., Meredith, G., eds.: COORDINATION. Volume 2949 of LNCS., Springer (2004) 87–104
9. Guidi, C., Lanese, I., Montesi, F., Zavattaro, G.: On the interplay between fault handling and request-response service invocations. Technical report (2007)
10. Butler, M.J., Hoare, C.A.R., Ferreira, C.: A trace semantics for long-running transactions. In Abdallah, A.E., Jones, C.B., Sanders, J.W., eds.: 25 Years Communicating Sequential Processes. Volume 3525 of LNCS., Springer (2004) 133–150
11. Bruni, R., Melgratti, H.C., Montanari, U.: Theoretical foundations for compensations in flow composition languages. In Palsberg, J., Abadi, M., eds.: POPL, ACM (2005) 209–220
12. Bruni, R., Melgratti, H.C., Montanari, U.: Nested commits for mobile calculi: Extending join. In Lévy, J.J., Mayr, E.W., Mitchell, J.C., eds.: IFIP TCS, Kluwer (2004) 563–576
13. Laneve, C., Zavattaro, G.: Foundations of web transactions. In Sassone, V., ed.: FoSSaCS. Volume 3441 of LNCS., Springer (2005) 282–298
14. Mazzara, M., Lanese, I.: Towards a unifying theory for web services composition. In Bravetti, M., Núñez, M., Zavattaro, G., eds.: WS-FM. Volume 4184 of LNCS., Springer (2006) 257–272
15. Vaz, C., Ferreira, C., Ravara, A.: Dynamic recovering of long running transactions. Technical report, CITI http://ctp.di.fct.unl.pt/˜cf/pub/DCpi.pdf.
16. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I and II. Inf. Comput. **100**(1) (1992) 1–77
17. Sangiorgi, D., Walker, D.: The π-calculus: a Theory of Mobile Processes. Cambridge University Press (2001)
18. Fournet, C., Gonthier, G.: The reflexive cham and the join-calculus. In: POPL. (1996) 372–385
19. Garcia-Molina, H., Salem, K.: Sagas. In Dayal, U., Traiger, I.L., eds.: SIGMOD Conference, ACM Press (1987) 249–259
20. Guidi, C., Lucchi, R., Gorrieri, R., Busi, N., Zavattaro, G.: : A calculus for service oriented computing. In Dan, A., Lamersdorf, W., eds.: ICSOC. Volume 4294 of LNCS., Springer (2006) 327–338

A Subject reduction

Lemma 1. *Let E be an runtime process. Then,* $\mathsf{fti}(\mathsf{extr}(E)) \subseteq \mathsf{fti}(E)$.

A straightforward induction on E.

Lemma 2. *If* $\Gamma \vdash P$ *then* $\Gamma \vdash \varphi(P)$

Simple, by induction on P.

Lemma 3. *If* $\Gamma \vdash P$ *then* $\mathsf{fti}(\{\varphi(P)\}) = \emptyset$

Simple, by induction on P.

Lemma 4. *Let* Γ *be a type environment and* E *a runtime process. If* $\Gamma \vdash a(\tilde{x})\,[E]$ *then* $\Gamma, \tilde{x} : \tilde{V} \vdash \mathsf{extr}(E)$.

A straightforward induction on E.

Lemma 5. *If* $E \longrightarrow E'$ *then* $\mathsf{fti}(E') \subseteq \mathsf{fti}(E)$.

Simply, by induction on the derivation $E \longrightarrow E'$.

Lemma 6. (Strengthening) *Let* Γ *be a type environment,* P *a compensation process and* E *an execution process.*

1. *If* $\Gamma, y : S \vdash P$ *and* y *is not free in* P *then* $\Gamma \vdash P$.
2. *If* $\Gamma, y : S \vdash E$ *and* y *is not free in* E *then* $\Gamma \vdash E$.

The proof is by induction on the typing derivations.

Lemma 7. (Weakening) *Let* Γ *be a type environment,* P *a compensation process and* E *an execution process.*

1. *If* $\Gamma \vdash P$ *then* $\Gamma, y : S \vdash P$ *for any type* S *and any name* y *which* Γ *is not defined.*
2. *If* $\Gamma \vdash E$ *then* $\Gamma, y : S \vdash E$ *for any type* S *and any name* y *which* Γ *is not defined.*

The proof is by induction on the typing derivations.

Lemma 8. (Substitution) *Let* Γ *be a type environment,* P *a compensation process and* E *an execution process.*

1. *If* $\Gamma \vdash P$, $\Gamma \vdash x : S$ *and* $\Gamma \vdash v : S$. *Then* $\Gamma \vdash P\{v/x\}$.
2. *If* $\Gamma \vdash E$, $\Gamma \vdash x : S$ *and* $\Gamma \vdash v : S$. *Then* $\Gamma \vdash E\{v/x\}$.

The proof is by induction on the typing derivations.

Lemma 9. (Structural Congruence) *Let* Γ *be a type environment,* E *and* D *execution processes. If* $\Gamma \vdash E$ *and* $E \equiv D$ *then* $\Gamma \vdash D$.

The proof is by induction on the derivation of $E \equiv D$ with a case-analysis on the last rule used. The inductive cases are the congruence rules, and are straightforward. Among the other cases, we show for axiom $a(\tilde{x})\,[E \mid \langle PP \rangle] \equiv a(\tilde{x})\,[E] \mid \langle PP \rangle$, in both directions.

- (Left-to-right). By assumption we have that $\Gamma \vdash a(\tilde{x})\,[E \mid \langle PP \rangle]$. From rule TE-SCOPE we can infer that

$$\Gamma, \tilde{x} : \tilde{V} \vdash E \mid \langle PP \rangle\,, \qquad \Gamma \vdash a : tc(\tilde{V}), \qquad a \notin \mathsf{fti}(E \mid \langle PP \rangle),$$

$$\tilde{x} \notin \mathsf{fen}(E \mid \langle PP \rangle), \qquad \text{and} \qquad \forall_{c \in \mathsf{ti}(E \mid \langle PP \rangle)}\ |c| = 0.$$

  From TE-PAR we can infer that

$$\Gamma, \tilde{x} : \tilde{V} \vdash E, \qquad \Gamma, \tilde{x} : \tilde{V} \vdash \langle PP \rangle \qquad \text{and} \qquad \mathsf{fti}(E) \cap \mathsf{fti}(\langle PP \rangle) = \emptyset.$$

  Since $\tilde{x} \notin \mathsf{fen}(E \mid \langle PP \rangle)$, by definition of free execution names we can conclude that $\tilde{x} \notin \mathsf{fen}(E)$ and $\tilde{x} \notin \mathsf{fen}(\langle PP \rangle)$. Therefore and by the fact that all compensation process defined in $\langle PP \rangle$ are the inaction process, $\tilde{x}$ is not free in $\langle PP \rangle$. Thus by Strengthening $\Gamma \vdash \langle PP \rangle$. we can also conclude that $a \notin \mathsf{fti}(E)$ since $a \notin \mathsf{fti}(E \mid \langle PP \rangle)$. From the fact that for all c in $\mathsf{ti}(E \mid \langle PP \rangle)$ we have $|c| = 0$ and from definition of free transaction identifiers, we can conclude that for all c in $\mathsf{ti}(E)$ we have $|c| = 0$. Thus, $\Gamma \vdash a(\tilde{x})\,[E]$. From rule TE-PAR we can infer that $\Gamma \vdash a(\tilde{x})\,[E] \mid \langle PP \rangle$.
- (Right-to-left). Suppose that $\Gamma \vdash a(\tilde{x})\,[E] \mid \langle PP \rangle$. From rule TE-PAR we can infer that

$$\Gamma \vdash a(\tilde{x})\,[E]\,, \qquad \Gamma \vdash \langle PP \rangle \qquad \text{and} \qquad \mathsf{fti}(a(\tilde{x})\,[E]) \cap \mathsf{fti}(\langle PP \rangle) = \emptyset$$

  From rule TE-SCOPE we can conclude that

$$\Gamma, \tilde{x} : \tilde{V} \vdash E, \qquad \Gamma \vdash a : tc(\tilde{V}), \qquad a \notin \mathsf{fti}(E), \qquad \tilde{x} \notin \mathsf{fen}(E), \qquad \text{and}$$

$$\forall_{c \in \mathsf{ti}(E)}\ |c| = 0.$$

  Therefore, Γ is not defined in $\tilde{x}$ and by Weakening Theorem we can conclude that $\Gamma, \tilde{x} : \tilde{V} \vdash \langle PP \rangle$ and that $\tilde{x} \notin \mathsf{fen}(\langle PP \rangle)$. From definition of free transaction identifiers and by the fact that $\mathsf{fti}(a(\tilde{x})\,[E]) \cap \mathsf{fti}(\langle PP \rangle) = \emptyset$, we can conclude that $\mathsf{fti}(E \cap \langle PP \rangle) = \emptyset$. Since $\langle PP \rangle$ does not have free transaction identifiers, we can conclude from TE-SCOPE that $\Gamma \vdash (\tilde{x})\,[E \mid \langle PP \rangle]$.

Proof of Theorem 1 (Subject Reduction)
By induction on the derivation of the reduction $E \longrightarrow E'$.

1. (R-COM): in this case,

$$E = \overline{a_j}\langle \tilde{v} \rangle \% Q \mid \sum_{i \in I} a_i(\tilde{x}_i) \% P_i . PP_i,$$

$$E' = \{\varphi(Q)\} \mid \{\varphi(P_j\{{}^{\tilde{v}}/{}_{\tilde{x}_j}\})\} \mid PP_j\{{}^{\tilde{v}}/{}_{\tilde{x}_j}\}$$

and by hypothesis, $j \in I$ and $|\tilde{v}| = |\tilde{x}_j|$. Since $\Gamma \vdash E$ we can infer from rule TE-PAR that

$$\Gamma \vdash \overline{a_j}\langle\tilde{v}\rangle\%Q, \qquad \Gamma \vdash \sum_{i\in I} a_i(\tilde{x_i})\%P_i.PP_i,$$

$$\mathsf{fti}(\overline{a_j}\langle\tilde{v}\rangle\%Q) \cap \mathsf{fti}(\sum_{i\in I} a_i(\tilde{x_i})\%P_i.PP_i) = \emptyset.$$

Using rules TE-INP and TE-OUT-C, we can conclude that

$$\Gamma, \tilde{x}_j : \tilde{V_j} \vdash PP_j, \qquad \Gamma \vdash a_j : ch(\tilde{V_j}), \qquad \Gamma, \tilde{x}_j : \tilde{V_j} \vdash P_j \qquad \text{and} \qquad \Gamma \vdash \tilde{v} : \tilde{V},$$

respectively.
By Lemma 7 (Weakening), $\Gamma, \tilde{x_j} \vdash \tilde{v}$. We can therefore apply the Substitution Lemma and infer

$$\Gamma, \tilde{x}_j : \tilde{V_j} \vdash PP_j\{^{\tilde{v}}/_{\tilde{x}_j}\}.$$

Since $\tilde{x}_j$ is not free in $PP_j\{^{\tilde{v}}/_{\tilde{x}_j}\}$ by Lemma 6 we can infer $\Gamma \vdash PP_j\{^{\tilde{v}}/_{\tilde{x}_j}\}$. Also, from rule TE-OUT-C, we can conclude that $\Gamma \vdash Q$. By Lemma 2 we have that $\Gamma \vdash \varphi(Q)$ and then by rule TE-STORED, $\Gamma \vdash \{\varphi(Q)\}$. Similarly and using Substitution Lemma and Lemma 6 we can also conclude that $\Gamma \vdash \{\varphi(P_j\{^{\tilde{v}}/_{\tilde{x}_j}\})\}$. By Lemma 3, we have

$$\mathsf{fti}(\{\varphi(Q)\}) = \emptyset \qquad \text{and} \qquad \mathsf{fti}(\{\varphi(P_j\{^{\tilde{v}}/_{\tilde{x}_j}\})\}) = \emptyset.$$

Thus, by rule TE-PAR

$$\Gamma \vdash \{\varphi(P_j\{^{\tilde{v}}/_{\tilde{x}_j}\})\} \mid \{\varphi(Q)\}$$

Again by rule TE-PAR, $\Gamma \vdash E'$.

2. (R-REP). Let

$$E = \overline{a_j}\langle\tilde{v}\rangle\%Q | !a(\tilde{x})\%P.PP,$$

$$E' = \{\varphi(Q)\} \mid \{\varphi(P)\} \mid PP\{^{\tilde{v}}/_{\tilde{x}}\} | !a(\tilde{x})\%P.PP.$$

and by hypothesis, $|\tilde{v}| = |\tilde{x}|$. Since $\Gamma \vdash E$ we can infer from rule TE-PAR that

$$\Gamma \vdash \overline{a_j}\langle\tilde{v}\rangle\%Q \qquad \text{and} \qquad \Gamma \vdash !a(\tilde{x})\%P.PP$$

Thus, we can infer by rule TE-REP that

$$\Gamma \vdash a : ch(\tilde{V}), \qquad \Gamma, \tilde{x} : \tilde{V} \vdash PP, \qquad \Gamma, \tilde{x} : \tilde{V} \vdash P \qquad \text{and} \qquad \mathsf{fti}(PP) = \emptyset.$$

From rule TE-OUT-C

$$\Gamma \vdash \tilde{v} : \tilde{V} \qquad \text{and} \qquad \Gamma \vdash Q.$$

By Lemma 7 (Weakening), $\Gamma, \tilde{x} \vdash \tilde{V}$. We can therefore apply the Substitution lemma 8 and infer

$$\Gamma, \tilde{x} : \tilde{V} \vdash PP\{\tilde{v}/\tilde{x}\}.$$

Since $\tilde{x}$ is not free in $PP\{\tilde{v}/\tilde{x}\}$, by Lemma 6, $\Gamma \vdash PP\{\tilde{v}/\tilde{x}\}$. Since $\Gamma \vdash Q$ we can conclude by Lemma 2 that $\Gamma \vdash \varphi(Q)$. Thus, by rule TE-STORED, $\Gamma \vdash \{\varphi(Q)\}$. Similarly and using Substitution Lemma and Lemma 6, we can conclude that $\Gamma \vdash \{\varphi(P\{\tilde{v}/\tilde{x}\})\}$. By Lemma 3

$$\mathsf{fti}(\{\varphi(Q)\}) = \emptyset \qquad \text{and} \qquad \mathsf{fti}(\varphi(P\{\tilde{v}/\tilde{x}\})) = \emptyset.$$

We can infer by rule TE-PAR that

$$\Gamma \vdash \{\varphi(Q)\} \mid \{\varphi(P\{\tilde{v}/\tilde{x}\})\}.$$

Since we have $\mathsf{fti}(PP) = \emptyset$ we can conclude by TE-PAR that

$$\Gamma \vdash PP\{\tilde{v}/\tilde{x}\} \,|!a(\tilde{x})\%P.PP.$$

Again, by TE-PAR, we have that $\Gamma \vdash E'$.

3. (R-RECOVER-OUT). In this case

$$E = \overline{a}\langle\tilde{v}\rangle\%P \mid a(\tilde{x})\,[D]\,, \qquad E' = \langle(\nu\,\tilde{y})\,(\prod_{i=0}^{n} QQ_i\{\tilde{v}/\tilde{x}\}) \mid \prod_{j=0}^{m}\langle RR_j\rangle\rangle \mid \{\varphi(P)\}$$

with

$$\mathsf{extr}(D) \equiv (\nu\,\tilde{y})\,(\prod_{i=0}^{n} QQ_i \;\mid\; \prod_{j=0}^{m}\langle RR_j\rangle).$$

Since $\Gamma \vdash E$ we can infer from rule TE-PAR that

$$\Gamma \vdash \overline{a}\langle\tilde{v}\rangle\%P \qquad \text{and} \qquad \Gamma \vdash a(\tilde{x})\,[D]\,.$$

Also, from TE-OUT-T we can conclude that

$$\Gamma \vdash \tilde{v} : \tilde{V}, \qquad \Gamma \vdash a : ch(\tilde{V}), \qquad \text{and} \qquad \Gamma \vdash P.$$

By Lemma 2, $\Gamma \vdash \varphi(P)$. Thus, by rule TE-STORED, $\Gamma \vdash \{\varphi(P)\}$. Also, by Lemma 3, $\mathsf{fti}(\{\varphi(P)\}) = \emptyset$. From Lemma 4, we can conclude that $\Gamma, x : \tilde{V} \vdash \mathsf{extr}(D)$. Thus, by Structural Congruence Lemma,

$$\Gamma, x : \tilde{V} \vdash (\nu\,\tilde{y})\,(\prod_{i=0}^{n} QQ_i \;\mid\; \prod_{j=0}^{m}\langle RR_j\rangle).$$

Using rule TE-SCOPE, $x \notin \mathsf{fen}(D)$ because by hypothesis $\Gamma \vdash a(\tilde{x})\,[D]$. Therefore, x cannot appear in any protected block subterm of D. The thesis follows using rules TE-RES, TE-PAR, Substitution Lemma and TE-BLOCK.

4. (R-RECOVER-IN) The proof is similar to the previous case.

5. (R-STRUCT) Let $E = F'$ and $E' = D'$. By hypothesis we have that

$$\Gamma \vdash F', \qquad F' \equiv F, \qquad F \rightarrow D, \qquad \text{and} \qquad D \equiv D'.$$

By Structural Congruence Lemma we have that $\Gamma \vdash F$ and therefore, by induction hypothesis, $\Gamma \vdash D$. Finally, by Structural Congruence Lemma $\Gamma \vdash D'$

6. (R-PAR-T). In this case, $E = F \mid D$, $E' = F' \mid D$ and by hypotheses we have that

$$F \longrightarrow F', \qquad \mathsf{ti}(D) \cap \mathsf{rdy}(D) = \emptyset \qquad \text{and} \qquad \Gamma \vdash F \mid D.$$

From rule TE-PAR we can infer that

$$\Gamma \vdash F \qquad \Gamma \vdash D \qquad \text{and} \qquad \mathsf{fti}(F) \cap \mathsf{fti}(D) = \emptyset.$$

By induction hypothesis, $\Gamma \vdash F'$ and by Lemma 5 we know that $\mathsf{fti}(F') \subseteq \mathsf{fti}(F)$. Thus, we can infer from rule TE-PAR that $\Gamma \vdash F' \mid D$.

7. (R-PAR-C) This proof is similar to the previous one.

8. (R-RES) In this case, $E = (\nu\, x)\, D$ and $E' = (\nu\, x)\, D'$. By hypothesis, we have that $\Gamma \vdash (\nu\, x)\, D$ and $D \rightarrow D'$. From TE-RES we can infer that $\Gamma, x : V \vdash D$. By induction hypothesis, $\Gamma, x : V \vdash D'$. The thesis follows using the rule TE-RES.

9. (R-SCOPE) Let

$$E = a(\tilde{x})\,[D] \qquad \text{and} \qquad E' = a(\tilde{x})\,[D']$$

By hypothesis we have that

$$\Gamma \vdash a(\tilde{x})\,[D] \qquad \text{and} \qquad D \rightarrow D'$$

From rule TE-SCOPE we can infer that

$$\Gamma, \tilde{x} : \tilde{V} \vdash D, \qquad \Gamma \vdash a : \mathsf{fn}(\tilde{V}), \qquad a \notin \mathsf{fti}(D), \qquad \tilde{x} \notin \mathsf{fen}(D) \qquad \text{and}$$

$$\forall_{c \in \mathsf{ti}(D)}\ |c| = 0$$

By induction hypothesis, we have that $\Gamma, \tilde{x} : \tilde{V} \vdash D'$. From Lemma 5 we can conclude that $a \notin \mathsf{fti}(D')$. Since $\tilde{x} \notin \mathsf{fen}(D)$, $\tilde{x}$ cannot be involved in any reduction from D to D'. Therefore, $\tilde{x} \notin \mathsf{fen}(D')$. Since for all $c \in \mathsf{ti}(D)$, $|c| = 0$, then we also have that for all $c \in \mathsf{ti}(D')$, $|c| = 0$. By TE-SCOPE the thesis follows.

10. (R-SCOPE-COM). In this case,

$$E = a(\tilde{x})[\,F\,] \mid D \qquad \text{and} \qquad E' = a(\tilde{x})[\,F'\,] \mid D'$$

Note that by hypothesis we have that $F \mid D \longrightarrow F' \mid D'$. TE-PAR gives

$$\Gamma \vdash a(\tilde{x})[\,F\,] \qquad \Gamma \vdash D \qquad \text{and} \qquad \mathsf{fti}(\,a(\tilde{x})[\,F\,]) \cap \mathsf{fti}(D) = \emptyset.$$

From rule TE-scope we can conclude that

$$\Gamma, \tilde{x} : \tilde{V} \vdash F, \qquad \Gamma \vdash a : tc(\tilde{V}), \qquad a \notin \mathsf{fti}(F), \qquad \tilde{x} \notin \mathsf{fen}(F), \qquad \text{and}$$

$$\forall_{c \in \mathsf{ti}(F)} \ |c| = 0$$

By Weakening, $\Gamma, \tilde{x} : \tilde{V} \vdash D$ and from rule TE-par we can infer $\Gamma, \tilde{x} : \tilde{V} \vdash F \mid D$. Thus by induction hypothesis, $\Gamma, \tilde{x} : \tilde{V} \vdash F' \mid D'$. Again, from TE-par, we can infer that

$$\Gamma, \tilde{x} : \tilde{V} \vdash F', \qquad \Gamma, \tilde{x} : \tilde{V} \vdash D' \qquad \text{and} \qquad \mathsf{fti}(F') \cap \mathsf{fti}(D') = \emptyset.$$

As $a \notin \mathsf{fti}(F)$, by Lemma 5 we know that $a \notin \mathsf{fti}(F')$. Thus, we can conclude from rule TE-scope that $\Gamma \vdash a(\tilde{x})\,[F']$. Also, since for all $c \in \mathsf{ti}(F)$, $|c| = 0$, then we also have that for all $c \in \mathsf{ti}(F')$, $|c| = 0$. As $\tilde{x}$ does not occur in Γ and $\tilde{x} \notin \mathsf{fen}(F)$, then $\tilde{x}$ is not free in D'. Therefore, by Strengthening Lemma, $\Gamma \vdash D'$. The thesis holds by TE-par.

11. (R-block) In this case,

$$E = \langle PP \rangle \qquad \text{and} \qquad E' = \langle PP' \rangle\,.$$

By hypothesis we have that

$$\Gamma \vdash \langle PP \rangle \qquad \text{and} \qquad PP \rightarrow PP'.$$

From rule TE-block we can infer that $\Gamma \vdash PP$. By induction hypothesis we have that $\Gamma \vdash PP'$. The thesis follows using the rule TE-block

12. (R-block-com) Let

$$E = \langle PP \rangle \mid D \qquad \text{and} \qquad E' = \langle PP' \rangle \mid D'$$

By hypothesis,

$$\Gamma \vdash \langle PP \rangle \mid D \qquad \text{and} \qquad PP \mid D \rightarrow PP' \mid D'.$$

From rule TE-par we can infer that $\Gamma \vdash \langle PP \rangle$, $\Gamma \vdash D$ and $\mathsf{fti}(\langle PP \rangle) \cap \mathsf{fti}(D) = \emptyset$. Rule TE-block gives $\Gamma \vdash PP$. By definition of free transaction names, $\mathsf{fti}(\langle PP \rangle) = \mathsf{fti}(PP)$. Thus, by Te-par, $\Gamma \vdash PP|D$. By induction hypothesis, $\Gamma \vdash PP' \mid D'$. From rule TE-par we can infer that

$$\Gamma \vdash PP', \qquad \Gamma \vdash D' \qquad \text{and} \qquad \mathsf{fti}(PP') \cap \mathsf{fti}(D') = \emptyset.$$

Rule TE-block gives $\Gamma \vdash \langle PP' \rangle$. Again, by definition of free transaction names, $\mathsf{fti}(\langle PP' \rangle) = \mathsf{fti}(PP')$. Therefore, $\Gamma \vdash E'$.

13. (R-recover-scope) Let

$$E = a(\tilde{x})\,[D \mid b(\tilde{z})\,[F]] \qquad E' = D'$$

Note that by hypothesis we have that

$$\Gamma \vdash E, \qquad a(\tilde{x})\,[D \mid F] \rightarrow D' \qquad \text{and} \qquad a \in \mathsf{rdy}(F).$$

Rule TE-SCOPE gives

$$\Gamma, \tilde{x} : \tilde{V} \vdash D \mid b(\tilde{z})\,[F]\,, \qquad \Gamma \vdash a : tc(\tilde{V}), \qquad a \notin \mathsf{fti}(D \mid b(\tilde{z})\,[F])$$

$$\tilde{x} \notin \mathsf{fen}(D \mid b(\tilde{z})\,[F]) \qquad \text{and} \qquad \forall_{c \in \mathsf{ti}(D \mid b(\tilde{z})\,[F])}\ |c| = 0.$$

Then $\tilde{z}$ is an empty tuple. We can infer from rule TE-PAR that

$$\Gamma, \tilde{x} : \tilde{V} \vdash D \qquad \Gamma, \tilde{x} : \tilde{V} \vdash b(\tilde{z})\,[F] \qquad \mathsf{fti}(D) \cap \mathsf{fti}(b(\tilde{z})\,[F]) = \emptyset$$

By rule TE-SCOPE we have that

$$\Gamma, \tilde{x}, \tilde{z} \vdash F, \qquad \Gamma, \tilde{x} \vdash b : \mathsf{ti}(\tilde{V}), \qquad b \notin \mathsf{fti}(F), \qquad \tilde{z} \notin \mathsf{fen}(F) \qquad \text{and}$$

$$\forall_{c \in \mathsf{ti}(F)}\ |c| = 0$$

Since $\tilde{z}$ is an empty tuple then $\Gamma, \tilde{x} \vdash F$. From rule TE-PAR and also by definition of free transaction identifiers we have that $\Gamma, \tilde{x} \vdash D \mid F$. Since $a \notin \mathsf{fti}(D \mid b(\tilde{z})\,[F])$ and $a \neq b$ then $a \notin \mathsf{fti}(D \mid F)$. Also, since $\tilde{x} \notin \mathsf{fen}(D \mid b(\tilde{z})\,[F])$ then $\tilde{x} \notin \mathsf{fen}(D \mid F)$. From the fact that for all $c \in \mathsf{ti}(D \mid b(\tilde{z})\,[F])$, $|c| = 0$, we can conclude that for all $c \in \mathsf{ti}(D \mid F)$, $|c| = 0$. Therefore, $\Gamma \vdash a(\tilde{x})\,[D \mid F]$. By induction hypothesis, $\Gamma \vdash D'$.

B Soundness and Type Safety

Proof of Theorem 2 (Soundness)

The proof that $\mathsf{in}(E) \cap \mathsf{ti}(E) = \emptyset$ for a runtime process E is an easy induction on the typing derivations.

The proof that given a runtime process E, $\mathsf{unq}(E)$ holds is by induction on the typing derivations.

- (TE-NIL), (TE-OUT-T) and (TE-OUT-C) rules. These cases are hold by definition of the predicate.
- (TE-INP) rule. Suppose that $\Gamma \vdash \sum_{i \in I} a_i(\tilde{x}_i)\%Q_i.PP_i$. From rule TE-INP, we know that$\Gamma, \cup_i\{x_i\} : \tilde{V}_i \vdash PP_i$ for all $i \in I$. Thus, by induction hypothesis, for all $i \in I$, $\mathsf{unq}(PP_i)$ holds which, by definition of the predicate, implies that $\mathsf{unq}(E)$ holds.
- (TE-REP) rule. In this case, E has the form $!a(\tilde{x}\%Q).PP$. From rule TE-REP we infer that $\mathsf{fti}(PP) = \emptyset$ and then the thesis follows.
- (TE-STORED) and (TE-BLOCK) rules. These cases are hold by definition of the predicate.
- (TE-RES) rule. In this case, $E = (\nu x)\ D$. From TE-RES rule $\Gamma, x : V \vdash D$. By induction hypothesis, $\mathsf{unq}(D)$ holds and so, by definition of the predicate, $\mathsf{unq}(E)$ holds.
- (TE-PAR) rule. E has the form $D \,|\, F$. From TE-PAR rule we infer that $\mathsf{fti}(D) \cap \mathsf{fti}(F) = \emptyset$, $\Gamma \vdash F$ and $\Gamma \vdash D$. By induction hypothesis, $\mathsf{unq}(F)$ and $\mathsf{unq}(D)$ holds. Therefore, by definition of the predicate, $\mathsf{unq}(E)$ holds.

- (TE-SCOPE) rule. Suppose that $E = a(\tilde{x})[D]$. From TE-SCOPE, $\Gamma \vdash D$ and $a \notin \mathsf{fti}(E)$. By induction hypothesis, $\mathsf{unq}(D)$ holds. Thus, by definition of the predicate, we can conclude that $\mathsf{unq}(E)$ holds.

As mentioned, a process is said to be well typed if it respects arity, *i.e.*, not fails. Definition 8 formalizes what we mean by (runtime) failure in our calculus.

Definition 8. *A* Runtime Failure *is inductively defined as:*

$$\frac{|\tilde{v}| \neq |\tilde{x}_j| \quad j \in I}{\overline{a}_j\,\langle\tilde{v}\rangle\,\%Q \mid \sum_{i\in I} a_i(\tilde{x}_i)\%P_i.PP_i\ fails} \qquad \frac{|\tilde{v}| \neq |\tilde{x}|}{\overline{a}\,\langle\tilde{v}\rangle\,\%Q|!a(\tilde{x})\%P.PP\ fails}$$

$$\frac{|\tilde{z}| \neq |\tilde{x}|}{\overline{a}\,\langle\tilde{z}\rangle\,\%P \mid a(\tilde{x})\,[E]\ fails} \qquad \frac{|\tilde{z}| \neq |\tilde{x}|}{a(\tilde{x})\,[E \mid \overline{a}\,\langle\tilde{z}\rangle\,\%P]\ fails} \qquad \frac{E\ fails}{(\nu\,x)\,E\ fails}$$

$$\frac{E \equiv D \quad D\ fails}{E\ fails} \qquad \frac{E\ fails}{a(\tilde{x})\,[E]\ fails} \qquad \frac{PP\ fails}{\langle PP\rangle\ fails} \qquad \frac{E\ fails}{E \mid D\ fails}$$

Fig. 8. Definition of runtime failure.

Lemma 10. *If $\Gamma \vdash E$ then E not fails.*

Suppose that E fails. We use induction on the depth of the inference of E fails to show that if $\Gamma \vdash E$ then we will have a contradiction.We show some cases, the others are similar.

- Case $E = \overline{a}_j\,\langle\tilde{v}\rangle\,\%Q \mid \sum_{i\in I} a_i(\tilde{x}_i)\%P_i.PP_i$ fails since $|\tilde{v}| \neq |\tilde{x}_j|$ with $j \in I$. Suppose that $\Gamma \vdash E$. Therefore, by TE-PAR we have that
$$\Gamma \vdash \overline{a}_j\,\langle\tilde{v}\rangle\,\%Q, \qquad \text{and} \qquad \Gamma \vdash \sum_{i\in I} a_i(\tilde{x}_i)\%P_i.PP_i$$
From rule TE-INP $\Gamma \vdash a_j(\tilde{x}_j)\%P_j.PP_j$. We can conclude from rules TE-OUT-C and TE-INP that the type of a_j is the same, so $|\tilde{v}| = |\tilde{x}_j|$ and we have a contradiction, as required.
- Case $E = a(\tilde{x})\,[D \mid \overline{a}\,\langle\tilde{z}\rangle\,\%P]$ fails since $|\tilde{z}| \neq |\tilde{x}|$. Suppose that $\Gamma \vdash E$. From rule TE-SCOPE we can infer that $\Gamma \vdash a : tc(\tilde{V})$ and $\Gamma, \tilde{x} : \tilde{V} \vdash D \mid \overline{a}\,\langle\tilde{z}\rangle\,\%P$. Since from rule TE-PAR we have that $\Gamma, \tilde{x} : \tilde{V} \vdash \overline{a}\,\langle\tilde{z}\rangle\,\%P$, we can conclude from rule TE-OUT-F and by the fact that $\Gamma \vdash a : tc(\tilde{V})$ that the type of a must be the same. Therefore, $|\tilde{v}| = |\tilde{x}_j|$ and we have a contradiction, as required.
- Case $E = (\nu\,x)\,D$ fails if D fails. Suppose that $\Gamma \vdash (\nu\,x)\,D$. From rule TE-RES we can infer that $\Gamma, x : S \vdash D$. By induction hypothesis, D not fails and we have a contradiction, as required.

- Case $E = \langle PP \rangle$ fails if PP fails. Suppose that $\Gamma \vdash \langle PP \rangle$. From rule TE-STORED $\Gamma \vdash PP$. By induction hypothesis, PP not fails and we have a contradiction, as required.

Proof of Theorem 3 (Type Safety) The proof is standard. It follows by the Lemma 10 and Subject Reduction Theorem.

C Properties of the recover mechanism

Lemma 11. *$E \equiv C[\![\bullet]\!]$ iff exists a context $C'[\![\bullet]\!]$ such that $E = C'[\![\bullet]\!]$.*

The proof is by induction on the number n of axioms of structural congruence applied. If we prove the thesis for $n = 1$ then the thesis in the general case follows by induction. For the base case, it is necessary to prove the thesis for each axiom. All the axioms can be applied either to the subterms of the productions denoted by E in the Definition 7 or to the operators. In the first case the thesis follows trivially since it is enough to choose an enabling context with a suitable E' instead of E. For the second case,the proof is done for each possible axiom of structural congruence applied. We will only show some cases, the other are similar.

- Commutative of $|$: this can only be applied to cases $C[\![\bullet]\!] \mid E$ and $E \mid C[\![\bullet]\!]$, and this corresponds to use the other case.
- $a(\tilde{x})[\,(\nu\, y)\, E\,] \equiv (\nu\, y)\,(a(\tilde{x})\,[E])\quad \text{if}\quad y \notin \tilde{x}$: this corresponds to exchange the order of two subsequent steps in building the context that use the cases $(\nu\, x)\, C[\![\bullet]\!]$ and $a(\tilde{x})\,[C[\![\bullet]\!]]$.
- $a(\tilde{x})\,[E \mid \langle PP \rangle] \equiv a(\tilde{x})\,[E] \mid \langle PP \rangle$: this case can be subdivided in two cases. One of them corresponds to exchange the order of two subsequent steps in building the context that use the cases $C[\![\bullet]\!] \mid \langle PP \rangle$ and $a(\tilde{x})\,[C[\![\bullet]\!]]$. In the other case, the building of a new context can be done with three subsequent steps $\langle C[\![\bullet]\!] \rangle$, $E \mid C[\![\bullet]\!]$ and $a(\tilde{x})\,[C[\![\bullet]\!]]$ or with two subsequent steps $\langle C[\![\bullet]\!] \rangle$ and $a(\tilde{x})\,[E] \mid C[\![\bullet]\!]$.
- $\langle PP \mid QQ \rangle \equiv \langle PP \rangle \mid \langle QQ \rangle$: this case can be subdivided in two cases. One of them the building of a new context can be done with two subsequent steps $C[\![\bullet]\!] \mid QQ$ and $\langle C[\![\bullet]\!] \rangle$ or with the subsequent steps $\langle C[\![\bullet]\!] \rangle$ and $C[\![\bullet]\!] \mid \langle QQ \rangle$. The other case is similiar.
- $\langle \mathbf{0} \rangle \equiv \mathbf{0}$: this cannot be applied outside de E subterms.

Lemma 12. *$E \equiv D[\![\bullet, \bullet]\!]$ iff exists a context $D'[\![\bullet, \bullet]\!]$ such that $E = D'[\![\bullet, \bullet]\!]$.*

The proof is by induction on the number n of axioms of structural congruence applied and is straightforward from the above Lemma.

Proof of Proposition 1

The proof is on the structure of $C[\![\bullet]\!]$. Notice that by Lemma 11 we have no need to consider structural congruence. We show some cases, the other are similar.

- $C[\![\bullet]\!] = \bullet$. Since $a \in \mathsf{rdy}(E)$, there is only one derivation rule that can be applied, the rule R-RECOVER-IN. Thus, the thesis holds.
  $C[\![\bullet]\!] = (\nu x)\ C'[\![\bullet]\!]$. The reduction can only be derived using rule RE-RES, and the thesis follows by inductive hypothesis.
- $C[\![\bullet]\!] = C'[\![\bullet]\!] \mid D'$. Since $a \in \mathsf{rdy}(E)$ and $a \in \mathsf{ti}(E)$, then $\mathsf{rdy}(D') \cap \mathsf{ti}(D') = \emptyset$. Thus, the only rule that can by applied is R-PAR-F and so the thesis follows by inductive hypothesis.
- $C[\![\bullet]\!] = D' \mid C'[\![\bullet]\!]$ It follows from Lemma 11 and the rule above.
- $C[\![\bullet]\!] = b(\tilde{y})\,[C'[\![\bullet]\!]]$. Notice that by the typing system $b \neq a$. In this case we must start by applying the reduction rule R-SCOPE and so the thesis follows by inductive hypothesis.
- $C[\![\bullet]\!] = \langle C[\![\bullet]\!]\rangle$. In this case $E = a(\tilde{x})\,[F \mid \langle C[\![\overline{a}\,\langle \tilde{y}\%Q\rangle]\!]\rangle]$. Note that in this case Q must be the inaction process. By structural congruence and Lemma 11, we only have to prove the thesis for $E^* = a(\tilde{x})\,[F] \mid \langle C[\![\overline{a}\,\langle \tilde{y}\%\mathbf{0}\rangle]\!]\rangle$. Since $\mathsf{ti}(E) \cap \mathsf{rdy}(E) = \{a\}$ then $\mathsf{ti}(E^*) \cap \mathsf{rdy}(E^*) = \{a\}$. Therefore the only rules that can be applied are R-BLOCK-COM and R-RECOVER-OUT. Thus $E' = \langle F'\rangle \mid \langle C[\![\{\mathbf{0}\}]\!]\rangle$ with$F' = (\nu\,\tilde{y})\,(\Pi_{i\in I} QQ_i\{^{\tilde{y}}/_{\tilde{x}}\} \mid \Pi_{j\in J}\langle RR_j\rangle)$ and $\mathsf{extr}(F) \equiv (\nu\,\tilde{y})\,(\Pi_{i\in I} QQ_i \mid \Pi_{j\in J}\langle RR_j\rangle)$. Since $\langle F'\rangle \mid \langle C[\![\{\mathbf{0}\}]\!]\rangle \equiv \langle F' \mid C[\![\{\mathbf{0}\}]\!]\rangle$ from Lemma 11 we can conclude that there is a context C' such that $E' = C'[\![\{\mathbf{0}\}]\!]$. Note that in this case we can define $C'[\![\bullet]\!]$ as $\langle F' \mid C[\![\bullet]\!]\rangle$. As $\{\mathbf{0}\} \equiv \mathbf{0}$, from Lemma 11 we can conclude that there is a context C'' such that $E' = C''[\![\mathbf{0}]\!]$. Note that we can define C'' as C'. The thesis follows considering the axiom $\mathbf{0} \equiv \mathbf{0}\%\mathbf{0}$ and from Lemma 11.

Proof of Proposition 2

The proof is on the structure of $D[\![\bullet,\bullet]\!]$. Since $D[\![\bullet,\bullet]\!] = C[\![\bullet]\!] \mid C'[\![\bullet]\!]$ all possible combinations of the structure of the contexts $C[\![\bullet]\!]$ and $C'[\![\bullet]\!]$ must be considered.
We show some cases, the others are similar.

- $D[\![\bullet,\bullet]\!] = \bullet \mid \bullet$. Since $a \in \mathsf{rdy}(E)$, there is only one derivation rule that can be applied, the rule R-RECOVER-OUT.
- $D[\![\bullet,\bullet]\!] = \bullet \mid b(\tilde{z})\,[C[\![\bullet]\!]]$. In this case $E = a(\tilde{x})\,[F] \mid b(\tilde{z})\,[C[\![\overline{a}\,\langle\tilde{y}\rangle\,\%Q]\!]]$. By Strutural Congruence and Lemma 12 we only have to prove the thesis for $b(\tilde{z})\,[C[\![\overline{a}\,\langle\tilde{y}\rangle\,\%Q]\!]] \mid a(\tilde{x})\,[F]$. Notice that by the typing system, $b \neq a$. Therefore, b is not ready to communicate and the only rule that can be applied is R-RECOVER-SCOPE. Thus by induction hypothesis and R-recover-scope, the thesis holds.
- $D[\![\bullet,\bullet]\!] = b(\tilde{z})\,[C_1[\![\bullet]\!]] \mid \langle C_2[\![\bullet]\!]\rangle$. Since in this case $E = b(\tilde{z})\,[C_1[\![a(\tilde{x})\,[F]]\!]] \mid \langle C_2[\![\overline{a}\,\langle\tilde{y}\rangle\,\%Q]\!]\rangle$ and $b \neq a$ by the type system, b is not ready to communicate. Therefore, the only rules that can be applied are R-SCOPE-COM and R-BLOCK-COM. Thus, by induction hypothesis and subsequently by rules R-SCOPE-COM and R-BLOCK-COM, the thesis holds.

Proof of Proposition 3

The proof is on the structure of $D[\![\bullet,\bullet]\!]$. Since $D[\![\bullet,\bullet]\!] = C[\![\bullet]\!] \mid C'[\![\bullet]\!]$ all possible combinations of the structure of the contexts $C[\![\bullet]\!]$ and $C'[\![\bullet]\!]$ must be

considered.
We show some cases, the other are similar.

- $D[\![\bullet,\bullet]\!] = \bullet \mid \bullet$.The thesis hold since there is only one derivation rule that can be applied, the rule R-COM. Thus, the thesis, follows.
- $D[\![\bullet,\bullet]\!] = \bullet \,|\, b(\tilde{z})\,[C[\![\bullet]\!]]$. In this case we have that $E = \sum_{i\in I} a_i(\tilde{x_i})\%P_i.PP_i \,|\, b(\tilde{z})\,[C[\![\overline{a}_j\,\langle\tilde{y}\rangle\,\%Q]\!]]$. By Strutural Congruence and Lemma 12 we only have to prove the thesis for $b(\tilde{z})\,[C[\![\overline{a}_j\,\langle\tilde{y}\rangle\,\%Q]\!]] \,|\, \sum_{i\in I} a_i(\tilde{x_i})\%P_i.PP_i$. Notice that by the typing system $b \neq a$. Therefore, b is not ready to communicate and the only rule that can be applied is R-SCOPE-COM. Thus by induction hypothesis and R-SCOPE-COM, the thesis holds.

Proof of Proposition 4

The proof is on the structure of $D[\![\bullet,\bullet]\!]$. Since $D[\![\bullet,\bullet]\!] = C[\![\bullet]\!] \mid C'[\![\bullet]\!]$ all possible combinations of the structure of the contexts $C[\![\bullet]\!]$ and $C'[\![\bullet]\!]$ must be considered.
The proof is on the structure of $D[\![\bullet,\bullet]\!]$. The proof is omitted since it is similar to the proof of the above definition.