Communicating Transactions*
(Extended Abstract)

Edsko de Vries, Vasileios Koutavas, and Matthew Hennessy

Trinity College Dublin
{Edsko.de.Vries,Vasileios.Koutavas,Matthew.Hennessy}@cs.tcd.ie

Dedicated to Robin Milner

Abstract. We propose a novel language construct called *communicating transactions*, obtained by dropping the isolation requirement from classical transactions, which can be used to model automatic error recovery in distributed systems. We extend CCS with this construct and give a simple semantics for the extended calculus, called TransCCS. We develop a behavioural theory which is sound and complete with respect to the may-testing preorder, and use it to prove interesting laws and reason compositionally about example systems. Finally, we prove that communicating transactions do not increase the observational power of processes; thus CCS equivalences are preserved in the extended language.

1 Introduction

Distributed systems such as web services [7] consist of a number of autonomous nodes in a network that communicate through message passing. As web services are increasingly designed by combining other web services through so-called *mashup* technologies [2], the complexity of these systems grows and *error recovery* becomes ever more difficult.

The usefulness of the transaction concept for the treatment of errors in such a setting has been recognized by both academia [19, 5] and industry [10, 12]. Error recovery in such transactions is based on *compensation*: services must programmatically bring the system back to a consistent state when an error has happened. In a distributed system of many independent components this may be difficult and error prone.

In many situations, however, automatic error recovery is possible through the use of classical techniques such as *rollback recovery* [16]. Processes store enough local state to be able to roll back after an error, and a rollback in one node may cause other nodes to rollback so that all nodes have a consistent view of the system state. The extent of the rollbacks can be limited through *coordinated checkpointing* [13], where processes coordinate to create a point beyond which they do not need to be rolled back.

In this paper we define a novel language construct of *communicating transactions*, which can be used to model the combination of rollback recovery and coordinated checkpointing. We give a high-level semantics of communicating

* This research was supported by SFI project SFI 06 IN.1 1898.

transactions in a calculus called TransCCS, an extension of CCS, and develop a compositional theory for this calculus based on may-testing equivalence.

Unlike traditional transactions, communicating transactions are not isolated: they may communicate with other processes or transactions in their environment. When a transaction communicates with its environment and subsequently fails, both the transaction and its environment will be rolled back to a consistent state. Transactions can commit to request a checkpoint; when transactions communicate, all must commit before the checkpoint is actually created (*cf.* the standard two-phase commit algorithm used for distributed transactions [27]).

In TransCCS we use $[\![P \triangleright_k Q]\!]$ to denote a transaction named k which currently runs P; a transaction is replaced by its *default* P after a commit, and by its *alternative* Q after an abort. *Restarting* transactions are modelled by recursive transactions $\mu X.[\![P \triangleright_k X]\!]$. A transaction can be aborted by the system at any point, and can commit using the language primitive $\mathtt{co}\ k$.

To give an intuition of communicating transactions we consider an informal semantics for communicating transactions. An example idealized execution of a system consisting of a merchant M (left) and a bank B (right) is given by:

$$
\begin{array}{lcl}
 & & req.\ [\![\tau.\overline{tr}.(\mathtt{co}\ k \mid \overline{ack}) \triangleright_k \overline{err}]\!] \mid \mu X.[\![tr.\tau.\mathtt{co}\ l \triangleright_l X]\!] \\
\text{(Place order)} & \xrightarrow{req} & [\![\tau.\overline{tr}.(\mathtt{co}\ k \mid \overline{ack}) \triangleright_k \overline{err}]\!] \mid \mu X.[\![tr.\tau.\mathtt{co}\ l \triangleright_l X]\!] \\
\text{(Process order)} & \xrightarrow{\tau} & [\![\ \overline{tr}.(\mathtt{co}\ k \mid \overline{ack}) \triangleright_k \overline{err}]\!] \mid \mu X.[\![tr.\tau.\mathtt{co}\ l \triangleright_l X]\!] \\
\text{(Request transfer)} & \xrightarrow{\tau} & [\![\ \ (\mathtt{co}\ k \mid \overline{ack}) \triangleright_k \overline{err}]\!] \mid [\![\ \tau.\mathtt{co}\ l \triangleright_l B]\!] \\
\text{(System aborts } l\text{)} & \xrightarrow{\tau} & \mid \mu X.[\![tr.\tau.\mathtt{co}\ l \triangleright_l X]\!] \qquad (1) \\
\text{(Cascading rollback)} & & [\![\ \overline{tr}.(\mathtt{co}\ k \mid \overline{ack}) \triangleright_k \overline{err}]\!] \mid \\
\text{(Second attempt)} & \xrightarrow{\tau}\xrightarrow{\tau} & [\![\ \ (\mathtt{co}\ k \mid \overline{ack}) \triangleright_k \overline{err}]\!] \mid [\![\ \mathtt{co}\ l \triangleright_l B]\!] \qquad (2) \\
\text{(Commit)} & \xrightarrow{\tau}\xrightarrow{\tau} & \overline{ack} \mid \mathtt{0} \qquad (3) \\
\text{(Acknowledge Order)} & \xrightarrow{\overline{ack}} & \mathtt{0} \mid \mathtt{0}
\end{array}
$$

In this trace M accepts an order on channel *req* and enters transaction k. Inside the transaction, M processes the order and issues a transfer request on tr to the bank B, which enters a (restarting) transaction l. The communication on tr should be considered tentative as it involves transactions k and l which are still subject to system failure. When the system decides to abort the l transaction in (1), it must also roll back the k transaction to a point before the transfer request in order to maintain global consistency; the k transaction, however, does not need to re-process the order. The second attempt to communicate between the transactions in (2) is also tentative, and only becomes a definitive action in (3) when both transactions have issued their commits. The acknowledgement of the order is then sent on *ack*. If at any point the system decides to abort the merchant transaction k (perhaps due to multiple failures to perform the transfer by the bank), an error signal is sent on *err*.

A direct formalization of this informal semantics would be quite complicated; for example dependencies between the various transactions would have to be maintained dynamically, and some notion of coordinated checkpointing or rollback would need to be implemented. In this paper we show that we can abstract away from such details through a simple concept called *embedding*. Specifically, we make the following contributions.

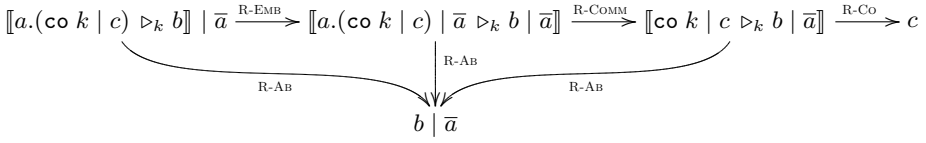
1. We give a simple reductions semantics for TransCCS by augmenting the standard semantics of CCS with a rule for *embedding* a process into a transaction, and two simple rules for committing and aborting transactions (Sect. 2).
2. We give a compositional behavioural theory for TransCCS (Sect(s). 3 to 5), based on *non prefix-closed* sets of traces derived by a Labelled Transition System (LTS), which is sound and complete with respect to *may-testing* [15]. The theory distinguishes between standard processes such as $a.b.\mathtt{0}$ in which all actions are definitive, and transactions $[\![a.b.\mathtt{co}\ k \rhd_k \mathtt{0}]\!]$ where the actions are tentative until transaction k commits.
3. We use the theory to prove a number of interesting laws about communicating transactions, including a theorem that transactions do not increase the observational power of processes and therefore CCS equivalences are preserved in TransCCS (Sect. 5). We also use the theory to reason compositionally about simple distributed systems (Sect. 6).
4. We study an extension to our calculus, TransCCS$^{\mu\mathtt{ab}}$, in which aborts are programmable (Sect. 7). We show that, provided all transactions are restarting, the characterization of may-testing in TransCCS is also valid in TransCCS$^{\mu\mathtt{ab}}$; we prove this through a simple fully abstract translation into TransCCS.

2 TransCCS

The syntax of TransCCS is that of CCS extended with a construct $[\![P \rhd_k Q]\!]$, denoting a transaction which is currently running its default P but which will be replaced by its alternative Q when it is aborted, and a construct $\mathtt{co}\ k$ to commit transactions, replacing it by its default.[1] The syntax and the reduction semantics are shown in Fig. 1; as usual a ranges over a set of actions Act on which is defined a bijective function $(\overline{\cdot}) : Act \to Act$, used to formalize communication, and μ ranges over Act_τ, the set Act augmented with a new action τ, used to represent internal activity. We use the standard abbreviations for CCS terms.

Although communication does not cross transaction boundaries, transactions can communicate through (non-deterministic) embedding:[2]

Example 1. Consider the reductions from a system consisting of the transaction $[\![a.(\mathtt{co} \mid c) \rhd_k b]\!]$ running in parallel with the simple process $\overline{a}$. Before communication can take place, the process must be embedded into the transaction; by embedding it into both the default and the alternative part of the transaction, we can restore the process to a consistent state after an abort. The possible traces are summarized in the graph below; note that a rollback (through R-Ab) remains possible until the commit has been executed. □



[1] After the commit, any remaining (possibly prefixed) $\mathtt{co}\ k$ statements behave like $\mathtt{0}$.
[2] Communication-driven embedding results in an equivalent but more complicated semantics.

Syntax

$$
\begin{array}{lll}
P, Q ::= \sum \mu_i.P_i \text{ guarded choice} & \mid [\![P \rhd_k Q]\!] & \text{transaction } (k \text{ bound in } P) \\
\mid (P \mid Q) \quad \text{parallel} & \mid \mathtt{co}\ k & \text{commit} \\
\mid \nu a.P \quad \text{hiding} & \mid \mu X.P & \text{recursion}
\end{array}
$$

Reduction Rules $(\rightarrow)$ is the least relation that satisfies

$$\text{R-Comm}\quad \frac{a_i = \overline{a}_j}{\sum_{i\in I} a_i.P_i \mid \sum_{j\in J} a_j.Q_j \rightarrow P_i \mid Q_j} \qquad \text{R-Emb}\quad \frac{k \notin R}{[\![P \rhd_k Q]\!] \mid R \rightarrow [\![P \mid R \rhd_k Q \mid R]\!]}$$

$$\text{R-Tau}\quad \frac{\mu_i = \tau}{\sum_{i\in I} \mu_i.P_i \rightarrow P_i} \qquad \text{R-Co}\quad \frac{}{[\![P \mid \mathtt{co}\ k \rhd_k Q]\!] \rightarrow P} \qquad \text{R-Ab}\quad \frac{}{[\![P \rhd_k Q]\!] \rightarrow Q}$$

$$\text{R-Rec}\quad \frac{}{\mu X.P \rightarrow P[X := \mu X.P]} \qquad \text{R-Str}\quad \frac{P \equiv P' \rightarrow Q' \equiv Q}{P \rightarrow Q}$$

and is closed under the contexts $C ::= [\,] \mid (C \mid Q) \mid [\![C \rhd_k Q]\!] \mid \nu a.C$.
Structural equivalence $(\equiv)$ contains the usual rules for parallel and hiding.

Fig. 1. Language Definition

Two or more transactions can communicate by taking mutual embedding steps, which is possible because transactions can be nested arbitrarily in TransCCS.[3]

Example 2. Consider again the Merchant M and Bank B transactions, together with a client $C = \overline{req}.P$. An example trace of $(C \mid M \mid B)$ is given by:

$$
\begin{aligned}
& \overline{req}.P \mid req.\,[\![\tau.\overline{tr}.(\mathtt{co}\ k \mid \overline{ack}) \rhd_k \overline{err}]\!] \mid \mu X.\,[\![tr.\tau.\mathtt{co}\ l \rhd_l X]\!] \\
\xrightarrow{\text{R-Comm}}\xrightarrow{\text{R-Tau}}\ & P \mid [\![\overline{tr}.(\mathtt{co}\ k \mid \overline{ack}) \rhd_k \overline{err}]\!] \mid \mu X.\,[\![tr.\tau.\mathtt{co}\ l \rhd_l X]\!] \\
\xrightarrow{\text{R-Emb}}\xrightarrow{\text{R-Rec}}\ & P \mid [\![\overline{tr}.(\mathtt{co}\ k \mid \overline{ack}) \mid [\![tr.\tau.\mathtt{co}\ l \rhd_l B]\!] \rhd_k \overline{err} \mid B]\!] \\
\xrightarrow{\text{R-Emb}}\ & P \mid [\![[\![\overline{tr}.(\mathtt{co}\ k \mid \overline{ack}) \mid tr.\tau.\mathtt{co}\ l \rhd_l \overline{tr}.(\mathtt{co}\ k \mid \overline{ack}) \mid B]\!] \rhd_k \ldots]\!] \\
\xrightarrow{\text{R-Comm}}\ & P \mid [\![[\![(\mathtt{co}\ k \mid \overline{ack}) \mid \tau.\mathtt{co}\ l \rhd_l \overline{tr}.(\mathtt{co}\ k \mid \overline{ack}) \mid B]\!] \rhd_k \overline{err} \mid B]\!] \qquad (*) \\
\xrightarrow{\text{R-Co}}\ & P \mid [\![(\mathtt{co}\ k \mid \overline{ack}) \rhd_k \overline{err} \mid B]\!] \xrightarrow{\text{R-Co}} P \mid \overline{ack}
\end{aligned}
$$

An alternative trace starting from $(*)$ begins with an abort of the bank:

$$
\begin{aligned}
& P \mid [\![[\![(\mathtt{co}\ k \mid \overline{ack}) \mid \tau.\mathtt{co}\ l \rhd_l \overline{tr}.(\mathtt{co}\ k \mid \overline{ack}) \mid B]\!] \rhd_k \overline{err} \mid B]\!] \\
\xrightarrow{\text{R-Ab}}\ & P \mid [\![\overline{tr}.(\mathtt{co}\ k \mid \overline{ack}) \mid B \rhd_k \overline{err} \mid B]\!]
\end{aligned}
$$

[3] We can embed B into M or vice-versa, but the two embeddings are equivalent.

The application of R-Ab also rolls back the merchant to before the output $\overline{tr}$; this is in conformance with the informal semantics in the introduction, as is the fact that the internal computation of the merchant is not rolled back. □

3 May Testing

We now apply the standard definition of may-testing to TransCCS. We will model a successful outcome of a test by a top-level output on a fresh channel ω.

Definition 1 (Barb). *$P\Downarrow_a$ iff there exist P_1 and P_2 such that $P \to^* P_1 \mid a.P_2$.*

Definition 2 (May-Testing Preorder). *We write $P \sqsubseteq_{\text{may}} Q$ iff for all processes T containing a fresh name ω, $(P \mid T)\Downarrow_\omega$ implies $(Q \mid T)\Downarrow_\omega$. We write $P \eqsim_{\text{may}} Q$ if $P \sqsubseteq_{\text{may}} Q$ and $Q \sqsubseteq_{\text{may}} P$.*

Example 3. Consider the systems $P_1 = [\![a.b.\mathtt{co}\ k \triangleright_k \mathbf{0}]\!]$, $P_2 = a.b$ and the test $T = \overline{a}.\omega$. When applied to P_2 the test succeeds since we reach the state $b \mid \omega$, but when applied to P_1 it fails since this leads to the failed state $[\![b.\mathtt{co}\ k \mid \omega \triangleright_k \mathbf{0} \mid \overline{a}.\omega]\!]$ (which does not have an ω-barb). Consequently $a.b \not\sqsubseteq_{\text{may}} [\![a.b.\mathtt{co}\ k \triangleright_k \mathbf{0}]\!]$. □

Our definition of barbs as top-level actions ensures that whenever $P\Downarrow_\omega$ then the action ω in P is definitive rather than tentative; the structure of processes in TransCCS ensures that top-level actions do not depend on the commitment of any transaction.[4] This is crucial to our notion of testing; for example the failed state above has the possibility of performing the action ω but this is tentative, as it depends on the transaction k committing. If k is aborted then this apparent success of the test would have to be rolled back.

In Sect. 5 we give a characterization of may-testing equivalence, with which we can give easy proofs for the following laws.

Proposition 1 (Uncommitted actions). *Actions within a transaction are not observable unless the transaction commits. For all P, Q, and R such that $k \notin R$ (in particular, $\mathtt{co}\ k \notin R$), we have*

$$[\![R \triangleright_k Q]\!] \eqsim_{\text{may}} Q \qquad \mu X.[\![R \triangleright_k X]\!] \eqsim_{\text{may}} \mathbf{0} \tag{1}$$

$$[\![P + R \triangleright_k Q]\!] \eqsim_{\text{may}} [\![P \triangleright_k Q]\!] \qquad \mu X.[\![P + R \triangleright_k X]\!] \eqsim_{\text{may}} \mu X.[\![P \triangleright_k X]\!] \tag{2}$$

Proposition 2 (Restarting transactions). $\mu X.[\![P \triangleright_k X]\!] \eqsim_{\text{may}} [\![P \triangleright_k \mathbf{0}]\!]$

Proposition 3 (Transactions versus processes).

$$[\![a.\mathtt{co}\ k \triangleright_k \mathbf{0}]\!] \eqsim_{\text{may}} a \qquad \mu X.[\![a.\mathtt{co}\ k \triangleright_k X]\!] \eqsim_{\text{may}} a \tag{3}$$

$$[\![P \mid \mathtt{co}\ k \triangleright_k \mathbf{0}]\!] \eqsim_{\text{may}} P \qquad \mu X.[\![P \mid \mathtt{co}\ k \triangleright_k X]\!] \eqsim_{\text{may}} P \tag{4}$$

$$[\![P \triangleright_k Q]\!] \sqsubseteq_{\text{may}} \tau.P + \tau.Q \qquad \mu X.[\![P \triangleright_k X]\!] \sqsubseteq_{\text{may}} P \tag{5}$$

[4] The theory of biorthogonality [24] yields the same barbs for our reduction semantics.

$$\text{L-Act}\ \frac{}{\sum \mu_i.P_i \xrightarrow{\mu_i} P_i} \qquad \text{L-Par}\ \frac{\mathcal{P} \xrightarrow{\widetilde{k}(\mu)} \mathcal{P}'}{\mathcal{P} \mid \mathcal{Q} \xrightarrow{\widetilde{k}(\mu)} \mathcal{P}' \mid \mathcal{Q}} \qquad \text{L-Trans}\ \frac{\mathcal{P} \xrightarrow{\widetilde{l}(\mu)} \mathcal{P}'}{[\![\mathcal{P} \triangleright_k \mathcal{Q}]\!] \xrightarrow{k(\widetilde{l}(\mu))} [\![\mathcal{P}' \triangleright_k \mathcal{Q}]\!]}$$

$$\text{L-Rec}\ \frac{}{\mu X.\mathcal{P} \xrightarrow{\tau} \mathcal{P}[X := \mu X.\hat{\mathcal{P}}]} \qquad \text{L-Hide}\ \frac{\mathcal{P} \xrightarrow{\mu} \mathcal{P}' \qquad a \notin \mu}{\nu a.\mathcal{P} \xrightarrow{\mu} \nu a.\mathcal{P}'} \qquad \text{L-Comm}\ \frac{\mathcal{P} \xrightarrow{\widetilde{k}(a)} \mathcal{P}' \qquad \mathcal{Q} \xrightarrow{\widetilde{k}(\overline{a})} \mathcal{Q}'}{\mathcal{P} \mid \mathcal{Q} \xrightarrow{\widetilde{k}(\tau)} \mathcal{P}' \mid \mathcal{Q}'}$$

(eliding L-Trans for secondary transactions)

Fig. 2. LTS: Standard Actions

4 Compositional LTS

4.1 Distributed Transactions

The use of the embedding rule R-Emb in the reduction semantics gives an easy to understand description of the execution of communicating transactions, but prevents compositional reasoning: parallel processes are no longer separate after embedding. For example, when trying to understand why the application of a test T to a system P is successful, embedding makes it difficult to disentangle the contributions made by T and P. After a number of reduction steps components of the test are distributed throughout the system, and vice-versa.

The LTS implements embedding differently. It is defined over an extended language, TransCCS$^\circ$, where transactions are distributed as a primary transaction, denoted by $[\![\mathcal{P} \triangleright_k \mathcal{Q}]\!]$, and zero or more secondary transactions, denoted by $[\![\mathcal{P} \triangleright_k \mathcal{Q}]\!]^\circ$. The system from Ex. 1 has the following trace in the LTS:

$$[\![a.(\mathtt{co}\ k \mid c) \triangleright_k b]\!] \mid \overline{a} \xrightarrow{\mathtt{emb}\ k} [\![a.(\mathtt{co}\ k \mid c) \triangleright_k b]\!] \mid [\![\overline{a} \triangleright_k \overline{a}]\!]^\circ \xrightarrow{k(\tau)} [\![\mathtt{co}\ k \mid c \triangleright_k b]\!] \mid [\![\mathbb{0} \triangleright_k \overline{a}]\!]^\circ \xrightarrow{\mathtt{co}\ k} c$$

The application of R-Emb is mimicked by the action $\mathtt{emb}\ k$ in the LTS, and the right process becomes a secondary k-transaction. The parallel composition of the primary transaction and secondary k-transaction should be thought of as modelling the transaction $[\![a.(\mathtt{co}\ k \mid c) \mid \overline{a} \triangleright_k b \mid \overline{a}]\!]$. The two processes remain separate, however, allowing for compositional reasoning. A step $\mathcal{P} \xrightarrow{\mathtt{emb}\ k} [\![\mathcal{P} \triangleright_k \mathcal{P}]\!]^\circ$ into a secondary k-transaction with no corresponding primary transaction models the embedding of $\mathcal{P}$ into a k transaction which is part of the environment.

The LTS is shown in Fig(s). 2 and 3. Judgements take the form:

- Communication actions $\mathcal{P} \xrightarrow{\widetilde{l}(\mu)} \mathcal{P}'$, which represent the tentative execution of μ inside the transactions $\widetilde{l}$.
- Broadcast actions $\mathcal{P} \xrightarrow{\beta} \mathcal{P}'$ where β can take the forms $(\mathtt{co}\ k)$ for committing, $(\mathtt{ab}\ k)$ for aborting, and $(\mathtt{emb}\ k)$ for embedding.

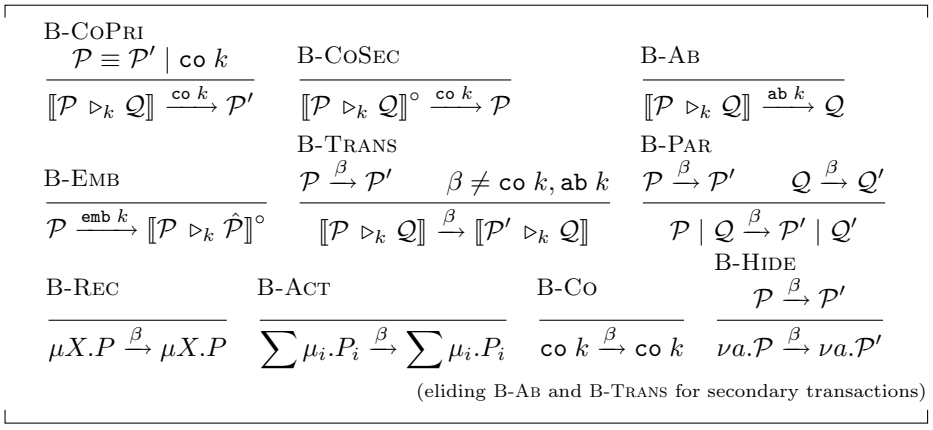


Fig. 3. LTS: Broadcast actions

We will refer to $\widetilde{k}(\tau)$ and broadcast actions as *silent* actions, and likewise to traces containing only silent actions as silent traces.

Communication actions are marked with their enclosing transactions (rule L-Trans). A $k(a)$ action can be matched by a parallel $k(\overline{a})$ action (L-Comm), modelling internal communication within the k-transaction.

When a primary k-transaction is ready to commit (B-CoPri), all secondary k-transactions must follow (B-CoSec). This is achieved by viewing the action $\mathtt{co}\ k$ as a broadcast action, which is propagated throughout the system (B-Par); non-transactions are unaffected by this action. Aborts are handled in a similar manner, although even primary transactions are subject to random system aborts (B-Ab). Embedding (B-Emb) is also a broadcast action to allow the distributed components of a process to be embedded simultaneously. Note that B-Hide does not require $a \notin \beta$ since we cannot restrict transaction names.

The rules in the LTS are subject to an implicit wellformedness condition, formally defined in [26], which guarantees that the distribution of transactions in a term indeed models a single transaction. For example, it prohibits terms such as

$$[\![[\![\mathcal{P}_1 \triangleright_k \mathcal{Q}_1]\!] \triangleright_l \ldots]\!]^\circ \mid [\![\mathcal{R} \triangleright_k \mathcal{R}]\!]^\circ \qquad \text{(illformed)}$$

The k-transaction cannot both be and not be embedded inside the l-transaction, and we therefore reject this term as illformed. Wellformedness also includes some technical but natural conditions that deal with freshness.

To support distribution, transactions are not binders in TransCCS$^\circ$ but are renamed when necessary (B-Emb, L-Rec) using an operation $\hat{\mathcal{P}}$. The implementation of $\hat{\mathcal{P}}$ is unimportant, but it must have the obvious properties (distribution over the constructors of the language, replacing names by sufficiently fresh ones, etc.), and defined so that two components of the same transaction (for instance, a primary and a secondary k-transaction) must be given the *same* new name. We use $\mathrm{tn}(\mathcal{P})$ to denote the set of names of the transactions in $\mathcal{P}$.

4.2 Relation to Reduction Semantics

To be able to formalize the relation between the reduction semantics and the LTS, we need to specify the mapping between terms in TransCCS$^\circ$ and TransCCS. We define an operation ($\rightsquigarrow$) which combines two k-transactions in a TransCCS$^\circ$ term into a single k-transaction.

Definition 3 (Merging). *($\rightsquigarrow$) is the least pre-congruence closed under structural equivalence that satisfies*

$$[\![\mathcal{P}_1 \rhd_k \mathcal{Q}_1]\!] \mid [\![\mathcal{P}_2 \rhd_k \mathcal{Q}_2]\!]^\circ \rightsquigarrow [\![\mathcal{P}_1 \mid \mathcal{P}_2 \rhd_k \mathcal{Q}_1 \mid \mathcal{Q}_2]\!]$$
$$[\![\mathcal{P}_1 \rhd_k \mathcal{Q}_1]\!]^\circ \mid [\![\mathcal{P}_2 \rhd_k \mathcal{Q}_2]\!]^\circ \rightsquigarrow [\![\mathcal{P}_1 \mid \mathcal{P}_2 \rhd_k \mathcal{Q}_1 \mid \mathcal{Q}_2]\!]^\circ$$

We use the symbol ($\leftrightsquigarrow$) for the symmetric closure of ($\rightsquigarrow$).

If we apply $\rightsquigarrow$ repeatedly, we eventually end up with a process with a single term for each transaction. If moreover the original process contained a primary k-transaction for every k (and not just secondary transactions), then we can regard the result as a TransCCS term. We overload $\mathcal{P} \rightsquigarrow P$ to describe this translation from a TransCCS$^\circ$ term to a TransCCS term.

We can now state that the LTS and the reduction semantics coincide:

Theorem 1 (Reduction semantics vs LTS). *Let $\mathcal{P} \rightsquigarrow P$.*

1. *If $P \to Q$ then there exist a process $\mathcal{Q}$ and silent action μ such that $\mathcal{P} \xrightarrow{\mu} \mathcal{Q}$ and $\mathcal{Q} \rightsquigarrow Q$. Moreover, if $\mu = \texttt{emb}\ k$ then $k \in \mathrm{tn}(\mathcal{P})$.*
2. *If $\mathcal{P} \xrightarrow{t} \mathcal{Q}$, where t is a silent trace, and whenever $\texttt{emb}\ k \in t$ then $k \in \mathrm{tn}(\mathcal{P})$, then there exist Q such that $P \to^* Q$ and $\mathcal{Q} \rightsquigarrow Q$.*

5 Characterization of May Testing

TransCCS encodes the complex interactions between communicating transactions. In this section we prove that the behaviour of transactional processes with respect to may-testing is characterized by a class of simple traces, which we call *clean traces*. We also prove that a weaker preorder which only uses non-transactional, sequential tests coincides with the may-testing preorder, and therefore CCS equivalences are preserved in TransCCS.

5.1 Clean Traces

Clean traces correspond to traces in the LTS where actions are never rolled back and are committed at the end of the trace: intuitively, every action in a clean trace eventually becomes definitive. Unlike LTS traces, however, clean traces do not include transaction names or broadcast actions. To enforce that all actions become definitive, the formal definition of clean traces (Fig. 4) is parametrized by a finite set of names Δ. Actions within a k-transaction can only occur in a clean trace if $k \in \Delta$ (C-Act, C-Emb), in which case k must commit at the end of the trace (C-Co) and cannot be aborted (C-Ab). We use $\texttt{co}\ \{k_1, \ldots, k_n\}$ for the process $(\texttt{co}\ k_1 \mid \cdots \mid \texttt{co}\ k_n)$.

$$\frac{\mathcal{P} \xrightarrow{\widetilde{k}(\mu)} \mathcal{P}'' \xrightarrow{t}_{\Delta} \mathcal{P}' \qquad \widetilde{k} \subseteq \Delta}{\mathcal{P} \xrightarrow{\mu,t}_{\Delta} \mathcal{P}'}\text{C-Act} \qquad \frac{\mathcal{P} \xrightarrow{\mathtt{ab}\ k} \mathcal{P}'' \xrightarrow{t}_{\Delta} \mathcal{P}' \qquad k \notin \Delta}{\mathcal{P} \xrightarrow{t}_{\Delta} \mathcal{P}'}\text{C-Ab}$$

$$\frac{\mathcal{P} \xrightarrow{\mathtt{emb}\ k} \mathcal{P}'' \xrightarrow{t}_{\Delta} \mathcal{P}' \qquad k \in \Delta}{\mathcal{P} \xrightarrow{t}_{\Delta} \mathcal{P}'}\text{C-Emb} \qquad \frac{\mathcal{P} \xrightarrow{\mathtt{co}\ \Delta} \mathcal{P}'}{\mathcal{P} \xrightarrow{\epsilon}_{\Delta} \mathcal{P}'}\text{C-Co}$$

Fig. 4. Clean Traces

Example 4. Let $\mathcal{P} = a.b + c$. As C-Act does not constrain top-level actions, the set of clean traces of $\mathcal{P}$ is $\{\epsilon, a, ab, c\}$ irrespective of the choice of Δ. □

Example 5. Let $\mathcal{P} = [\![a.b.\mathtt{co}\ k \triangleright_k c]\!]$. If we choose $\Delta = \{k\}$, we can only derive the clean trace ab:

$$\frac{[\![a.b.\mathtt{co}\ k \triangleright_k c]\!] \xrightarrow{k(a)} \quad \dfrac{[\![b.\mathtt{co}\ k \triangleright_k c]\!] \xrightarrow{k(b)} \dfrac{[\![\mathtt{co}\ k \triangleright_k c]\!] \xrightarrow{\mathtt{co}\ k} \mathbb{0}}{[\![\mathtt{co}\ k \triangleright_k c]\!] \xrightarrow{\epsilon}_{\{k\}} \mathbb{0}}\text{C-Co} \quad k \in \{k\}}{[\![b.\mathtt{co}\ k \triangleright_k c]\!] \xrightarrow{b}_{\{k\}} \mathbb{0}}\text{C-Act} \quad k \in \{k\}}{[\![a.b.\mathtt{co}\ k \triangleright_k c]\!] \xrightarrow{a,b}_{\{k\}} \mathbb{0}}\text{C-Act}$$

With this choice of Δ we cannot derive the empty trace because the k transaction is unable to commit immediately (nor can it be aborted). However, if we pick $\Delta = \emptyset$, we can derive the clean traces ϵ (using C-Co) and c (using C-Ab).

The singleton trace a is not derivable as a clean trace with *any* choice of Δ. As in the derivation above, we need $k \in \Delta$ to do a $k(a)$ action but the transaction is unable to commit until the b action. Clean traces are thus not prefix closed: $\mathcal{P}$ cannot do a definite a without also doing a definitive b.

Similarly, the trace abc is not derivable as a clean trace with any choice of Δ, because we need $k \in \Delta$ to do the $k(a)$ and $k(b)$ actions, and $k \notin \Delta$ to abort the k transaction and do the c action. $\mathcal{P}$ can *either* do a definitive a and b, *or* a definitive c, but not both. □

Normally the choice of Δ is not important:

Definition 4. *We write $\mathcal{P} \xrightarrow{t}_{CL}$ iff t is a clean trace of $\mathcal{P}$, that is $\exists \Delta, \mathcal{P}'$ such that $\mathcal{P} \xrightarrow{t}_{\Delta} \mathcal{P}'$. We write $\mathcal{P} \overset{t}{\Rightarrow}_{CL}$ to denote that t is a weak clean trace of $\mathcal{P}$.*[5]

Example 6. The set of clean traces of $[\![a.b.\mathtt{co}\ k \triangleright_k c]\!]$ (Ex. 5) is $\{\epsilon, ab, c\}$. □

Example 7. Let $\mathcal{P} = \nu a.\Big([\![a.b.\mathtt{co}\ k \triangleright_k \mathbb{0}]\!] \mid [\![\overline{a}.(\tau.\mathtt{co}\ l + \overline{b}) \triangleright_l \mathbb{0}]\!]\Big)$. The set of clean traces of $\mathcal{P}$ is $\{\epsilon, \tau b\}$. The trace $\tau\tau$ (internal communication on both a and b) is not derivable for any Δ because the l transaction cannot commit after doing a $\overline{b}$. The trace $\tau\overline{b}$ is not derivable for similar reasons. □

[5] Since clean traces are CCS traces, we can use the standard definition of a weak trace.

5.2 Soundness and Completeness

Our theory of may-testing is based on weak clean trace inclusion.

Definition 5 (Clean Trace Preorder). *We write* $\mathcal{P} \sqsubseteq_{\approx_{\mathrm{tr}}} \mathcal{Q}$ *iff* $\mathcal{P} \stackrel{t}{\Rightarrow}_{CL}$ *implies* $\mathcal{Q} \stackrel{t}{\Rightarrow}_{CL}$.

The clean trace preorder is sound and complete with respect to the may-testing preorder (Def. 2).

Theorem 2 (Soundness). *If* $P \sqsubseteq_{\approx_{\mathrm{tr}}} Q$ *then* $P \sqsubseteq_{\approx_{\mathrm{may}}} Q$.

Theorem 3 (Completeness). *If* $P \sqsubseteq_{\approx_{\mathrm{may}}} Q$ *then* $P \sqsubseteq_{\approx_{\mathrm{tr}}} Q$.

We now define a weaker testing preorder that uses only non-transactional, sequential tests, which coincides with the may-testing and clean trace preorders.

Definition 6 (Non-Transactional Testing Preorder). *We write* $\mathcal{P} \hat{\sqsubseteq}_{\approx_{\mathrm{may}}} \mathcal{Q}$ *iff for all tests* T *of the form* $a_1.a_2.\ldots.a_n.\omega$, $(P \mid T)\Downarrow_\omega$ *implies* $(Q \mid T)\Downarrow_\omega$.

Theorem 4 (Conservativity). $P \hat{\sqsubseteq}_{\approx_{\mathrm{may}}} Q$ *iff* $P \sqsubseteq_{\approx_{\mathrm{may}}} Q$.

The final theorem entails that equivalent CCS processes are also equivalent TransCCS processes; i.e. communicating transactions do not increase the distinguishing power of the language.

5.3 Proof Outline

The proof that may-testing is characterized by weak clean traces (Thm(s). 2 and 3) is a non-trivial result, and we can give but a sketch of the proof here. A more detailed proof can be found in a companion technical report [26].

For may-testing we are interested in (silent) traces that result in a top-level barb (ω). The first result states that whenever a process can ring with an arbitrary trace, it can ring with a clean trace:

Proposition 4. *Let* s *be a silent trace such that* $\mathcal{P} \stackrel{s}{\rightarrow} \mathcal{R} \mid \omega$. *Then there exists* Δ, *silent clean* t, *and* $\mathcal{R}'$ *such that* $\mathcal{P} \stackrel{t}{\rightarrow}_\Delta \mathcal{R}' \mid \omega$.

Proof (outline). First we inspect s and pick a Δ containing exactly the transactions that commit in s. Then we construct t by induction on s. Actions in s which happen inside transactions that do not commit (and are not in Δ) cannot contribute to the ring and are simply skipped. Similarly, if a transaction aborts we make sure to abort it before any other action inside the transaction. Finally, we delay all commits to the end of the trace. The final process $\mathcal{R}'$ may differ from the final process after the original trace $\mathcal{R}$, but it will ring. As a simple example, the trace

$$[\![\tau.\mathbf{0} \rhd_k \mathbf{0}]\!] \mid [\![\tau.\mathbf{0} \rhd_l [\![\tau.\omega \mid \mathtt{co}\ m \rhd_m \mathbf{0}]\!]]\!] \xrightarrow{k(\tau),l(\tau),\mathtt{ab}\ l,\mathtt{co}\ m,\tau} [\![\mathbf{0} \rhd_k \mathbf{0}]\!] \mid \omega$$

will be converted to the trace $\mathtt{ab}\ l, m(\tau), \mathtt{co}\ m$, used to derive the clean trace

$$[\![\tau.\mathbf{0} \rhd_k \mathbf{0}]\!] \mid [\![\tau.\mathbf{0} \rhd_l [\![\tau.\omega \mid \mathtt{co}\ m \rhd_m \mathbf{0}]\!]]\!] \stackrel{\tau}{\rightarrow}_{\{m\}} [\![\tau.\mathbf{0} \rhd_k \mathbf{0}]\!] \mid \omega \qquad \square$$

We assume the standard definition of "zipping" of two CCS-like traces ($t \# t'$) that allows interleaving and communication between the actions of the traces [26]. The next result is crucial; it states that zipping is a meaningful operation on clean traces; i.e., that the parallel composition of two processes can do any clean trace in the zip of the clean traces of the individual processes.

Proposition 5 (Zipping). *Let* $\mathcal{P} \xrightarrow{t_1}_{\Delta} \mathcal{P}'$, $\mathcal{Q} \xrightarrow{t_2}_{\Delta} \mathcal{Q}'$ *and* $\mathrm{tn}(\mathcal{P}) \cap \mathrm{tn}(\mathcal{Q}) \subseteq \Delta$. *Then for all* $t \in t_1 \# t_2$ *there exists* $\mathcal{R}$ *such that* $\mathcal{P} \mid \mathcal{Q} \xrightarrow{t}_{\Delta} \mathcal{R} \leftrightsquigarrow (\mathcal{P}' \mid \mathcal{Q}')$.

Prop. 5 is an important but non-trivial result, which requires a proof that transaction structure does not limit communication. For example, let

$$[\![a.\mathtt{co}\ k \triangleright_k \mathbf{0}]\!] \xrightarrow{a}_{\{k,l\}} \mathbf{0} \qquad \text{and} \qquad [\![\overline{a}.\mathtt{co}\ k \triangleright_l \mathbf{0}]\!] \xrightarrow{\overline{a}}_{\{k,l\}} \mathbf{0}$$

Then the parallel composition has the trace

$$\begin{array}{rll} & [\![a.\mathtt{co}\ k \triangleright_k \mathbf{0}]\!] & \mid [\![\overline{a}.\mathtt{co}\ k \triangleright_l \mathbf{0}]\!] \\ \xrightarrow{\mathtt{emb}\ k} & [\![a.\mathtt{co}\ k \triangleright_k \mathbf{0}]\!] & \mid [\![[\![\overline{a}.\mathtt{co}\ k \triangleright_l \mathbf{0}]\!] \triangleright_k [\![\overline{a}.\mathtt{co}\ k \triangleright_l \mathbf{0}]\!]]\!]^\circ \\ \xrightarrow{\mathtt{emb}\ l} & [\![[\![a.\mathtt{co}\ k \triangleright_l a.\mathtt{co}\ k]\!]^\circ \triangleright_k \mathbf{0}]\!] & \mid [\![[\![\overline{a}.\mathtt{co}\ k \triangleright_l \mathbf{0}]\!] \triangleright_k [\![\overline{a}.\mathtt{co}\ k \triangleright_l \mathbf{0}]\!]]\!]^\circ \\ \xrightarrow{k(l(\tau)),\mathtt{co}\ l,\mathtt{co}\ k} & \mathbf{0} & \mid \mathbf{0} \end{array}$$

Hence we can derive the clean trace $[\![a.\mathtt{co}\ k \triangleright_k \mathbf{0}]\!] \mid [\![\overline{a}.\mathtt{co}\ k \triangleright_l \mathbf{0}]\!] \xrightarrow{\tau}_{\{k,l\}} \mathbf{0}$.

Proposition 6 (Completeness w.r.t. ($\hat{\sqsubseteq}_{\mathbf{may}}$)). *If* $P \hat{\sqsubseteq}_{\mathrm{may}} Q$ *then* $P \sqsubseteq_{\mathrm{tr}} Q$.

Proof. Standard, using Prop. 5 and an easy unzipping lemma.

Given Prop(s). 4 and 5 and Thm. 1, soundness (Thm. 2) can be proven in a standard way. Completeness (Thm. 3) and conservativity (Thm. 4) follow from Prop. 6 and soundness.

6 Examples

The soundness theorem means that we can prove may-testing equivalences based on weak clean trace inclusion. In this section we give a few examples.

Proposition 3(5). $[\![P \triangleright_k Q]\!] \sqsubseteq_{\mathrm{may}} \tau.P + \tau.Q$.

Proof (outline). Let $[\![P \triangleright_k Q]\!] \stackrel{t}{\Rightarrow}_{\mathrm{CL}}$, where $t = \mu_1, \ldots, \mu_n$. That is, $\exists \mathcal{R}, \Delta$ such that $[\![P \triangleright_k Q]\!] \xrightarrow{t}_{\Delta} \mathcal{R}$. Either $k \in \Delta$ or $k \notin \Delta$. If $k \in \Delta$ then t corresponds to a trace $k(\widetilde{l}_1(\mu_1))$, …, $k(\widetilde{l}_2(\mu_2))$, …, $\mathtt{co}\ k$ of actions inside k interspersed with broadcast actions and ending on a commit of k. This means that P will have a trace $\widetilde{l}_1(\mu_1)$, …, $\widetilde{l}_2(\mu_2)$, … which corresponds to the *same* clean trace t. On the other hand, if $k \notin \Delta$ then t is the empty trace, or it must correspond to a trace that starts with an abort of k, followed by a trace s of Q. Since the abort is not part of the clean trace, s corresponds to the same clean trace t. Hence $\tau.P + \tau.Q$ includes the weak clean traces of $[\![P \triangleright_k Q]\!]$. □

Example 8. $[\![a.b.\mathtt{co}\ k \triangleright_k c]\!] \sqsubseteq_{\approx_{\text{may}}} a.b + c$ but not vice versa.

Proof. The inclusion follows from 3(5). We can also prove it directly, because as we saw in Sect. 5.1, the set of clean traces of the former process is $\{\epsilon, a, ab, c\}$ while the set of clean traces of the latter is $\{\epsilon, ab, c\}$. □

Theorem 5 (Compositional reasoning). *If* $P \sqsubseteq_{\approx_{\text{tr}}} Q$ *then* $P \mid R \sqsubseteq_{\approx_{\text{tr}}} Q \mid R$.

Proof. Since may-testing supports compositional reasoning, this theorem follows directly from soundness and completeness. □

Example 9. Consider the following alternative implementation of the bank and merchant example from the introduction, in which the merchant M' tries to complete the order twice before reporting an error back to the client.

$$\mathit{req}.\ [\![\tau.\overline{tr}.(\mathtt{co}\ k \mid \overline{ack})\ \triangleright_k\ [\![\overline{tr}.(\mathtt{co}\ k \mid \overline{ack})\ \triangleright_k\ \overline{err}]\!]]\!] \mid \mu X.\ [\![tr.\tau.\mathtt{co}\ l\ \triangleright_l\ X]\!]$$

By compositional reasoning, we only need to prove the two implementations of the merchant equivalent ($M \approx_{\text{may}} M'$) to prove the two systems equivalent.

Intuitively, an observer cannot distinguish between M and M' because when either merchant aborts, the observer is also rolled back: aborts are not detectable. Moreover, the observable behaviour of M', before or after the first abort, equals the behaviour of M.

Formally, the set of weak clean traces of both implementations equals the set $\{\epsilon, \mathit{req}, \mathit{req}\,\overline{tr}, \mathit{req}\,\overline{tr}\,\overline{ack}, \mathit{req}\,\overline{err}\}$. □

7 Programmable Aborts

In TransCCS aborts are entirely non-deterministic. We now turn our attention to *programmable aborts*; i.e. aborts that are triggered by the process through a new language primitive $\mathtt{ab}\ k$. The semantics of this new construct is given by rule R-Prog-Ab, below, replacing rule R-Ab:

$$\text{R-Prog-Ab}\qquad \frac{}{[\![\mathtt{ab}\ k \mid P\ \triangleright_k\ Q]\!] \to Q}$$

This new language, called TransCCS$^{\mathtt{ab}}$, does not however preserve consistency after an abort. Programmable aborts introduce an undesirable causal dependency between the alternative behaviour of a transaction which follows the abort, and the actions that led to that abort. For example, after the reduction

$$a \mid [\![\overline{a}.\mathtt{ab}\ k\ \triangleright_k\ b.\omega]\!] \xrightarrow{\text{R-Emb}} \xrightarrow{\text{R-Comm}} \xrightarrow{\text{R-Prog-Ab}} a \mid \omega$$

a communication on channel b is available *because* a communication on channel a led to an abort; *but* this communication on a is undone. Hence, from the point of view of the left process a the communication has not yet happened, but from the point of view of the transaction it has and led to an abort.

This is more than just a philosophical objection: using transactions such as the above as tests we can show that the basic equivalences in Prop. 1 are not preserved in TransCCS$^{\mathtt{ab}}$. For example, take the two transactions

$$P = [\![b.\mathtt{co}\ l \triangleright_l \mathtt{0}]\!] \qquad Q = [\![b.\mathtt{co}\ l + a \triangleright_l \mathtt{0}]\!]$$

and the same test $T = [\![\overline{a}.\mathtt{ab}\ k \triangleright_k b.\omega]\!]$. Then P fails the test T but Q does not, and hence $P \not\approx_{\text{may}} Q$, *even though* the a action is never committed:

$$\begin{aligned}
&[\![b.\mathtt{co}\ l + a \triangleright_l \mathtt{0}]\!] \mid [\![\overline{a}.\mathtt{ab}\ k \triangleright_k b.\omega]\!] \\
\xrightarrow{\text{R-Emb}}\ &[\![(b.\mathtt{co}\ l + a) \mid [\![\overline{a}.\mathtt{ab}\ k \triangleright_k b.\omega]\!] \triangleright_l T]\!] \\
\xrightarrow{\text{R-Emb}}\ &[\![[\![(b.\mathtt{co}\ l + a) \mid \overline{a}.\mathtt{ab}\ k \triangleright_k (b.\mathtt{co}\ l + a) \mid b.\omega]\!] \triangleright_l T]\!] \\
\xrightarrow{\text{R-Comm}}\ &[\![[\![\mathtt{ab}\ k \triangleright_k (b.\mathtt{co}\ l + a) \mid b.\omega]\!] \triangleright_l T]\!] \\
\xrightarrow{\text{R-Prog-Ab}}\ &[\![(b.\mathtt{co}\ l + a) \mid b.\omega \triangleright_l T]\!] \xrightarrow{\text{R-Comm}} [\![\mathtt{co}\ l \mid \omega \triangleright_l T]\!] \xrightarrow{\text{R-Co}} \omega
\end{aligned}$$

We can recover the preservation of consistency, however, by restricting all transactions to be restarting, i.e. of the form $\mu X.[\![P \triangleright_k X]\!]$; we call this language TransCCS$^{\mu\mathtt{ab}}$. Unfortunately, it is difficult to reason directly about TransCCS$^{\mu\mathtt{ab}}$ processes, since the language is not closed under reduction: after a restarting transaction unfolds and its default process reduces, we are left with a TransCCS$^{\mathtt{ab}}$ transaction whose default and alternative processes are different. However, we can give a theory for TransCCS$^{\mu\mathtt{ab}}$ through a *fully-abstract translation* to TransCCS, and reason about the behaviour of TransCCS$^{\mu\mathtt{ab}}$ processes via the translation.

We consider the translation $\{\!|\cdot|\!\} : \text{TransCCS}^{\mu\mathtt{ab}} \to \text{TransCCS}$, which maps $\mathtt{ab}\ k$ to $\mathtt{0}$ and is the identity on all other constructs. We use the annotation "$\mu\mathtt{ab}$" to refer to the semantics of TransCCS$^{\mu\mathtt{ab}}$. The important property of the translation is that it preserves barbs. From that and the results of Sect. 5 we derive the theorems of soundness and full abstraction.

Proposition 7. $P{\Downarrow}^{\mu\mathtt{ab}}_{\omega}$ *iff* $\{\!|P|\!\}{\Downarrow}_{\omega}$.

Theorem 6 (Full Abstraction of $\{\!|\cdot|\!\}$). $P \sqsubseteq^{\mu\mathtt{ab}}_{\text{may}} Q$ *iff* $\{\!|P|\!\} \sqsubseteq_{\text{may}} \{\!|Q|\!\}$.

8 Related Work

To the extent of our knowledge there is little related work on modelling automatic error recovery of communicating systems. Most work has either focused on models for isolated transactions [4, 17], including software transactional memory [18, 1], or compensation-based transactions [5, 9, 8, 20, 11] where error recovery must be programmed explicitly.

TransCCS is motivated by the long literature on implementing distributed systems with automatic error recovery (e.g. [16, 13, 22, 23]) and their verification in process calculi (such as [3, 6, 21]). This work, however, is only indirectly related to ours as we are not proposing a mechanism for *implementing* automatic

rollback recovery but rather a way to give high-level specifications of, and reason about, distributed systems that rely on automatic error recovery.

The only other language that we are aware of in which non-isolated transactions can be modelled is Reversible CCS [14]. RCCS extends CCS with the notion of *reversible actions* (written $a, \overline{a}, \ldots$) and *irreversible actions* (written $\underline{a}, \underline{\overline{a}}, \ldots$) past which processes cannot be rolled back. This can be used to model simple (non-nested) transactions; for example, a transaction superficially similar to $\mu X.\, [\![a + b.\mathtt{co}\ k\ \triangleright_k\ X]\!]$ can be written as $(a + \underline{b})$ in RCCS. The dynamic behaviour of these two terms is significantly different however: the RCCS transaction is not in charge of when it commits. An irreversible action $\underline{\overline{a}}$ by an observer can interact with the reversible action a of the transaction, forcing the transaction to commit. Thus the test $\overline{a}.\underline{\omega}$ succeeds when paired with the above transaction in RCCS, but must fail in TransCCS. The same observer can distinguish $\mathtt{0}$ from the transaction $[\![a\ \triangleright_k\ \mathtt{0}]\!]$ (in our syntax), which are indistinguishable in TransCCS. Hence a testing theory of RCCS would have to take into account the non-committing traces of transactions, in contrast to our theory for TransCCS in which the behaviour of a transaction only depends on its committing behaviour.

9 Conclusions

We presented a novel language construct called communicating transactions, which makes it possible to describe the behaviour of distributed systems with automatic error recovery at a high level of abstraction. We believe that support for communicating transactions may be beneficial in the design and implementation of complex distributed systems such as web services.

We introduced TransCCS, an extension of CCS with this construct. To the extent of our knowledge TransCCS is the first calculus which encapsulates both rollback recovery and coordinated checkpointing. We gave simple semantics to TransCCS and developed a basic behavioural theory, based on non prefix-closed sets of traces, that characterizes may-testing. We used the theory to prove a number of interesting laws and reason compositionally about example systems. We also studied TransCCS$^{\mu\mathtt{ab}}$, a variant of the language with programmable aborts, and gave a fully-abstract translation to TransCCS.

We plan to study the must-testing or fair-testing [25] theory of TransCCS in order to be able to specify liveness properties in the presence of aborts; we expect that the translation from TransCCS$^{\mu\mathtt{ab}}$ into TransCCS from Sect. 7 will not be fully abstract with respect to these testing preorders. We also plan to extend our work to the π-calculus and other behavioural equivalences such as bisimulation. Finally, we intent to investigate the usefulness of the construct of communicating transactions in a more realistic programming language.

References

1. Acciai, L., Boreale, M., Zilio, S.D.: A concurrent calculus with atomic transactions. In: ESOP. LNCS, vol. 4421, pp. 48–63. Springer (2007)

2. Benslimane, D., Dustdar, S., Sheth, A.: Services mashups: The new generation of web applications. IEEE Internet Computing 12, 13–15 (2008)
3. Berger, M., Honda, K.: The two-phase commitment protocol in an extended π-calculus. In: EXPRESS. ENTCS, vol. 39, pp. 21–46. Elsevier (2003)
4. Black, A.P., Cremet, V., Guerraoui, R., Odersky, M.: An equational theory for transactions. In: FSTTCS. LNCS, vol. 2914, pp. 38–49. Springer (2003)
5. Bocchi, L.: Compositional nested long running transactions. In: FASE. LNCS, vol. 2984, pp. 194–208. Springer (2004)
6. Bocchi, L., Wischik, L.: A process calculus of atomic commit. In: WS-FM. ENTCS, vol. 105, pp. 119–132. Elsevier Science Publishers (2004)
7. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web services architecture (February 2004), W3C Working Group Note
8. Bruni, R., Melgratti, H., Montanari, U.: Theoretical foundations for compensations in flow composition languages. In: POPL. pp. 209–220. ACM (2005)
9. Bruni, R., Melgratti, H., Montanari, U.: Nested commits for mobile calculi: extending Join. In: IFIP-TCS. pp. 569–582. Kluwer Academic Publishers (2004)
10. Cabrera, L.F., et al.: Web services business activity framework (WS-BusinessActivity) (August 2005), Whitepaper
11. Caires, L., Ferreira, C., Vieira, H.T.: A process calculus analysis of compensations. In: TGC. LNCS, vol. 5474, pp. 87–103 (2008)
12. Ceponkus, A., Dalal, S., Fletcher, T., Furniss, P., Green, A., Pope, B.: Business transaction protocol (June 2002), OASIS Committee Specification
13. Chandy, K.M., Lamport, L.: Distributed snapshots: determining global states of distributed systems. ACM Trans. Comp. Syst. 3(1), 63–75 (1985)
14. Danos, V., Krivine, J.: Transactions in RCCS. In: CONCUR. LNCS, vol. 3653, pp. 398–412. Springer-Verlag (2005)
15. De Nicola, R., Hennessy, M.C.B.: Testing equivalences for processes. Theoretical Computer Science 34(1–2), 83–133 (1984)
16. Elnozahy, E.N.M., Alvisi, L., Wang, Y.M., Johnson, D.B.: A survey of rollback-recovery protocols in message-passing systems. ACM Comp. Surv. 34(3), 375–408 (2002)
17. Gorrieri, R., Marchetti, S., Montanari, U.: A^2CCS: atomic actions for CCS. Theor. Comp. Sci. 72(2-3), 203–223 (1990)
18. Harris, T., Marlow, S., Peyton-Jones, S., Herlihy, M.: Composable memory transactions. In: PPoPP. pp. 48–60. ACM (2005)
19. Little, M.: Transactions and web services. Commun. ACM 46(10), 49–54 (2003)
20. Lucchi, R., Mazzara, M.: A pi-calculus based semantics for WS-BPEL. Journal of Logic and Algebraic Programming 70(1), 96–118 (2007)
21. Nestmann, U., Fuzzati, R., Merro, M.: Modeling consensus in a process calculus. In: CONCUR. LNCS, vol. 2761, pp. 399–414. Springer (2003)
22. Nett, E., Mock, M.: How to commit concurrent, non-isolated computations. In: FTDCS. pp. 343–353. IEEE Comp. Soc. (1995)
23. Park, T., Lee, I., Yeom, H.Y.: An efficient causal logging scheme for recoverable distributed shared memory systems. Parallel Computing 28(11), 1549–1572 (2002)
24. Rathke, J., Sassone, V., Sobocinski, P.: Semantic barbs and biorthogonality. In: FoSSaCS. LNCS, vol. 4423, pp. 302–316. Springer-Verlag (2007)
25. Rensink, A., Vogler, W.: Fair testing. Inf. and Comp. 205(2), 125–198 (2007)
26. de Vries, E., Koutavas, V., Hennessy, M.: Communicating transactions—technical appendix (April 2010), available at `http://www.scss.tcd.ie/Edsko.de.Vries`
27. Weikum, G., Vossen, G.: Transactional information systems, chap. 20 (Distributed Transaction Recovery). Morgan Kaufmann Publishers Inc. (2001)