Liveness of 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

Abstract. We study liveness and safety in the context of CCS extended with *communicating transactions*, a construct we recently proposed to model automatic error recovery in distributed systems. We show that fair-testing and may-testing capture the right notions of liveness and safety in this setting, and argue that must-testing imposes too strong a requirement in the presence of transactions. We develop a sound and complete theory of fair-testing in terms of CCS-like tree failures and show that, compared to CCS, communicating transactions provide increased distinguishing power to the observer. We also show that weak bisimilarity is a sound, though incomplete, proof technique for both may- and fair-testing. To the best of our knowledge this is the first semantic treatment of liveness in the presence of transactions. We exhibit the usefulness of our theory by proving illuminating liveness laws and simple but non-trivial examples.

1 Introduction

The correctness of distributed systems can to a large extent be specified in terms of its safety and liveness properties. In the presence of some form of built-in fault tolerance, such as support for transactions, the verification of safety properties is simplified but the verification of liveness properties becomes more subtle.

In previous work [22] we defined the novel language construct of *communicating transactions*, which drops the isolation requirement of classical transactions and models automatic error recovery of distributed communicating systems. We gave a high-level semantics of communicating transactions in a calculus called TransCCS, an extension of CCS, and developed a compositional theory for this calculus based on may-testing preorder.

May-testing can be used to reason about safety [14]. The intuition of safety is that "nothing bad will happen" [18]. A safety property can be formulated as a safety test T^{ϖ} which detects and reports the bad behaviour on a channel ϖ. We say that a process P *passes* a safety test if $(P \mid T^{\varpi})$ cannot report on ϖ. An implementation I then preserves the safety properties of a specification S if I passes all the safety tests of S (i.e. $S \sqsupseteq_{\text{may}} I$).

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

Let us briefly consider in a value-passing version of TransCCS a simple distributed communication system *Sys* that implements the specification

$$Spec_{rec,del} = rec(x).\overline{del}\langle x\rangle.\mathbf{0}$$

The implementation uses restarting communicating transactions. A restarting transaction is written as $\mu X.[\![P \triangleright_k X]\!]$ and executes its *default* P until P commits the transaction by executing a $\mathtt{co}\ k$ or the runtime non-deterministically aborts the transaction. P can communicate with the environment of the transaction, but these effects are rolled back automatically in the case of an abort.

$$\begin{aligned}
Sys_{rec,del} &= \nu q.\nu s.\left(Src_{rec,s,q} \mid Trg_{del,s,q} \mid \overline{s}\langle 0\rangle\right)\\
Src_{rec,s,q} &= \mu X\,[\![s(x).\mathtt{if}\ x = 0\ \mathtt{then}\ rec(y).(\overline{q}\langle y\rangle \mid \overline{s}\langle 1\rangle \mid \mathtt{co}\ k)\ \mathtt{else}\ \mathbf{0} \triangleright_k X]\!]\\
Trg_{del,s,q} &= \mu X\,[\![s(x).\mathtt{if}\ x = 1\ \mathtt{then}\ q(y).(\overline{del}\langle y\rangle \mid \overline{s}\langle 0\rangle \mid \mathtt{co}\ l)\ \mathtt{else}\ \mathbf{0} \triangleright_l X]\!]
\end{aligned}$$

The system uses a one-place queue, with *Src* storing the value received on *rec* as an output on q, if the current size of the queue, stored in s, is 0; *Trg* conveys the value from q to $\overline{del}$, if the queue is not empty. Both *Src* and *Trg* rely on an abort to undo the input on s if their condition is not satisfied.

As discussed, *Sys* is a safe implementation of *Spec* if $Spec \sqsupseteq_{\text{may}} Sys$. This would guarantee that an observer testing for a violation of the safety property that the received and delivered values match,

$$T^{\varpi} = \overline{rec}\langle v\rangle.del(x).\mathtt{if}\ x = v\ \mathtt{then}\ \mathbf{0}\ \mathtt{else}\ \varpi$$

can not report ϖ with *Sys* because it can not report this with *Spec*.

The intuition of *liveness* is that "something good will eventually happen". As for safety properties, a liveness property can be formulated as a liveness test T^{ω} which detects and reports the good behaviour on a channel ω. For example,

$$T^{\omega} = \omega + \overline{rec}\langle v\rangle.del(x).\omega$$

tests for the property that *if* an input is received on *rec* we eventually get an output on *del* (ω appears twice in the test because the implication can be satisfied in two ways). The definition of *passing* a liveness test however is delicate.

One possibility, corresponding to *must-testing* [11], is to require that every computation of $(Sys \mid T^{\omega})$ reports success. A restarting transaction can however be aborted by the runtime system infinitely often, even though at every point in the computation the transaction can follow a path to a commit. Under such, admittedly pathological, schedules no restarting transaction can guarantee liveness: in $(Sys \mid T^{\omega})$, after the value is received on *rec*, infinite aborts of *Trg* will prevent the value from being delivered on *del*, and the test will not succeed along this schedule. Indeed, under this scheme there would be no difference between *Sys* and the process $\mu X.\tau.X$, as neither can guarantee any liveness properties.

A more useful definition assumes a notion of fairness, and considers only schedules where *every transaction that gets a chance to commit infinitely often will eventually do so* [8]. We say that a process P passes a liveness test T^{ω} if

$(P \mid T^\omega)$ *should* report success on ω: $(P \mid T^\omega)$ will eventually ring under a fair scheduler. This definition of passing a liveness test leads to fair-testing [21].

With that definition we can show that *Sys* is a live implementation of *Spec*: *Sys* passes all the liveness tests of *Spec* (i.e. $Spec \sqsubseteq_{\text{fair}} Sys$). This is a non-trivial property of *Sys* which implies, among others, that relying on the abort of the transactions to restore the output on s when the conditions of *Src* and *Trg* are not satisfied does not introduce any deadlocks.

We make the following contributions in this paper:

1. We study liveness and safety in a concurrent language with communicating transactions and show that these notions are captured respectively by fair-testing and may-testing. To the best of our knowledge this is the first semantic treatment of liveness in the presence of transactions.
2. We give a characterization of liveness preservation in TransCCS in terms of so-called *clean tree failures*. This builds on previous results about clean traces (traces that contain only actions that will be committed), as well as newly proved properties of communicating transactions and the identification of characteristic TransCCS liveness tests.
3. We show that transactions add observational power to the observer with respect to liveness preservation and explain this through examples.
4. We define a variation on weak bisimilarity over clean traces and show that this is a sound but incomplete proof technique for safety and liveness.
5. We exhibit the usefulness of our theory by illuminating laws and examples.

2 Syntax and Reduction Semantics of TransCCS

The syntax and the reduction semantics of TransCCS are shown in Fig. 2; 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.

TransCCS extends CCS with the constructs $[\![P \triangleright_k Q]\!]$ which denotes a transaction, and $\mathtt{co}\ k$ which commits transaction k. The transaction runs its *default* P, which replaces the transaction in the case of a commit. The *alternative* Q replaces the transaction in the case of a non-deterministic abort. The occurrences of k in P are bound by the transaction; after a commit any remaining free $\mathtt{co}\ k$ behave as the nil process. Fig. 1 shows some simple examples of transactions. We will refer to these examples throughout the paper.

$$S_{ab} = \mu X.\, [\![a.b.\mathtt{co}\ k \triangleright_k X]\!]$$

$$I_1 = [\![a.b.\mathtt{co}\ k \triangleright_k \mathtt{0}]\!] \qquad I_3 = \mu X.\, [\![a.b.\mathtt{co}\ k + \overline{err} \triangleright_k X]\!]$$

$$I_2 = \mu X.\, [\![a.b.\mathtt{0} \triangleright_k X]\!] \qquad I_4 = \mu X.\, [\![a.b.\mathtt{co}\ k \mid \overline{err} \triangleright_k X]\!]$$

Fig. 1. Example Transactions

Syntax

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

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

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

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

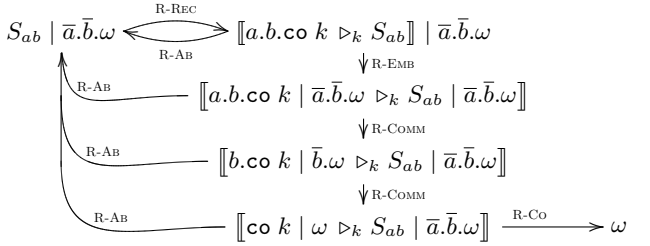
$$\text{R-Rec}\quad \frac{}{\mu X.P \rightarrow P[X := \mu X.P]} \qquad \text{R-Str}\quad \frac{P \equiv P' \rightarrow Q' \equiv Q}{P \rightarrow Q}$$

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

Fig. 2. Language Definition

A transaction can communicate with a process R in its environment by *embedding* R in its default and alternative (R-Emb). This simple but important operation allows the default of the transaction to interact with R.

Example 1. Consider transaction S_{ab} in parallel with the test $T^{\omega}_{ab} = \overline{a}.\overline{b}.\omega$. After an embedding step, the transaction can communicate with the process; both will be restored to their original state in the case of an abort. The possible traces are summarized in the following graph.



3 Liveness

We now formalize liveness as described in the introduction. A process P (typically the parallel composition of a test and a process-under-test) can output on a channel ω, written $P\Downarrow_{\omega}$ if it can reach a top-level ω after some internal steps:

Definition 1. *$P\Downarrow_{\omega}$ iff there exist P' such that $P \rightarrow^* \omega \mid P'$.*

We are interested only in top-level occurrences of ω, because an output that is still inside a transaction may still be undone.

A process P passes a liveness test T^ω if it cannot reach a state from which the test cannot detect the good behaviour.

Definition 2 (Passing Liveness Tests). *A process P passes a liveness test T^ω, written $P \,\mathrm{shd}\, T^\omega$, when $\forall R$. if $P \mid T^\omega \to^* R$ then $R \Downarrow_\omega$.*

Example 2. Transaction S_{ab} passes the liveness test $T^\omega_{ab} = \overline{a}.\overline{b}.\omega$ even though it may keep aborting. S_{ab} would *not* pass this test with the stronger definition of liveness (must-testing [11]), which would not ignore this pathological schedule.

The reduction graph of $S_{ab} \mid T^\omega_{ab}$ was shown in Ex. 1. Although there are infinite aborting paths, at no point does the system reach a state in which communication on a and b has become impossible. This is not true for I_1:

$$I_1 \mid T^\omega_{ab} \xrightarrow{\text{R-Ab}} \mathtt{0} \mid T^\omega_{ab} \not\Downarrow_\omega$$

Transaction I_2 also fails the liveness test since the transaction never commits and the output on ω by the test therefore never becomes top-level. □

Given a specification S, an implementation I preserves the liveness properties of S if every successful liveness test of S is also a successful liveness test of I. This naturally leads us to the standard definition of *fair-testing* [21] which here we call *liveness preservation.*

Definition 3 (Liveness Preservation). *I preserves the liveness properties of S, written $S \sqsubseteq_{\text{live}} I$, when for all liveness tests T^ω, if $S \,\mathrm{shd}\, T^\omega$ then $I \,\mathrm{shd}\, T^\omega$. We write $S \eqsim_{\text{live}} I$ if $S \sqsubseteq_{\text{live}} I$ and $I \sqsubseteq_{\text{live}} S$.*

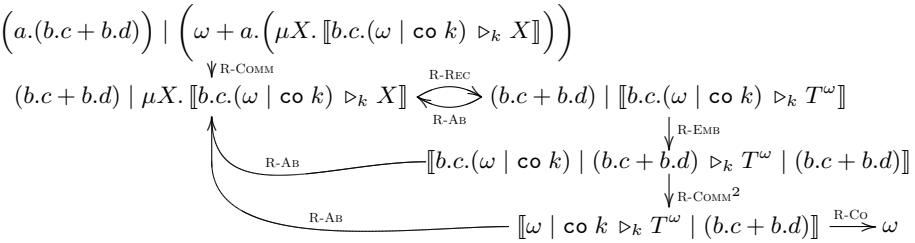
Example 3. We saw in Ex. 2 that S_{ab} passes the test T^ω_{ab} and that neither I_1 nor I_2 does. It follows immediately that $S_{ab} \not\sqsubseteq_{\text{live}} I_1$ and $S_{ab} \not\sqsubseteq_{\text{live}} I_2$. □

Example 4. We will formally prove $S_{ab} \sqsubseteq_{\text{live}} I_3$ after we develop our theory of liveness. Here we note only that $I_3 \,\mathrm{shd}\, T^\omega_{ab}$, which is easy to see from the reduction graph of $I_3 \mid T^\omega_{ab}$ (which is almost identical to the graph shown in Ex. 1). □

Example 5 (Transactional Liveness Tests). Another interesting case are the processes $P = a.(b.c + b.d)$ and $Q = a.b.c + a.b.d$. In TransCCS (unlike in CCS) we have that $P \not\sqsubseteq_{\text{live}} Q$. To see that consider the transactional liveness test

$$T^\omega = \omega + a.\Big(\mu X.\, [\![b.c.(\omega \mid \mathtt{co}\ k) \rhd_k X]\!]\Big)$$

We can see that P passes this test by the reduction graph of $P \mid T^\omega_{(a,bd)}$:



The restarting transaction makes it possible to restore the choice $b.c + b.d$ if the wrong branch of P communicates on b. However, Q does not pass this test:

$$(a.b.c + a.b.d) \mid \omega + a.\Big(\mu X.\, [\![b.c.(\omega \mid \mathtt{co}\ k)\ \triangleright_k\ X]\!]\Big)$$
$$\xrightarrow{\text{R-Comm}} b.d \mid \mu X.\, [\![b.c.(\omega \mid \mathtt{co}\ k)\ \triangleright_k\ X]\!] \not\Downarrow_\omega \qquad \square$$

Proposition 1. *If* $\mathtt{co}\ k \notin R$ *then*

$$[\![P + R \triangleright_k Q]\!] \eqsim_{\text{live}} [\![P \triangleright_k Q]\!] \qquad \mu X.\, [\![P + R \triangleright_k X]\!] \eqsim_{\text{live}} \mu X.\, [\![P \triangleright_k X]\!]$$
$$[\![P \triangleright_k Q]\!] \precsim_{\text{live}} \tau.P + \tau.Q \qquad \mu X.\, [\![P \mid \mathtt{co}\ k \triangleright_k X]\!] \eqsim_{\text{live}} P$$

In TransCCS, unlike CCS, the point of internal choice is important with respect to liveness preservation.

Proposition 2 (Choice). $\tau.a.P + \tau.a.Q \sqsubseteq_{\text{live}} a.P + a.Q \eqsim_{\text{live}} a.(\tau.P + \tau.Q)$.

To see that $\tau.a.P + \tau.a.Q \not\precsim_{\text{live}} a.P + a.Q$ consider the processes $R_1 = \tau.a.c + \tau.a.d$ and $R_2 = a.c + a.d$ and the liveness test $T^\omega = \mu X.\, [\![\overline{a}.\overline{c}.(\omega \mid \mathtt{co}\ k \triangleright_k X]\!])$. R_2 passes this test but R_1 does not (*cf.* Ex. 5).

Proposition 3 (Compositionality Laws). *If* $P \sqsubseteq_{\text{live}} P'$ *and* $Q \sqsubseteq_{\text{live}} Q'$ *then:*

$$a.P \sqsubseteq_{\text{live}} a.P' \qquad P \mid Q \sqsubseteq_{\text{live}} P' \mid Q' \qquad a.P + b.Q \sqsubseteq_{\text{live}} a.P' + b.Q'$$

These laws can be proven using our characterization of liveness preservation in Sect. 6. As in CCS [21], however, recursive contexts do *not* preserve ($\sqsubseteq_{\text{live}}$), as illustrated in the next example.

Example 6 (Fault tolerance). Consider the processes $P = a + \tau$ and $Q = \mathtt{0}$. P can be thought of as a process with a fault: it may do an a action or it may get stuck. Any liveness test that P passes can therefore not rely on the a action, and hence we have $P \sqsubseteq_{\text{live}} Q$. However, consider the context

$$C = \nu a.\mu X.\, [\![\overline{a}.(b \mid \mathtt{co}\ k) \mid [\,]\ \triangleright_k\ X]\!]$$

This context adds fault tolerance: if P faults in $C[P]$, the transaction can abort and try again, so that $C[P]$ will pass the liveness test $T_b^\omega = \overline{b}.\omega$. However, Q *never* does the a action, so the addition of fault tolerance makes no difference; in particular, $C[Q]$ does not pass T_b^ω. Hence, $C[P] \not\precsim_{\text{live}} C[Q]$. $\square$

4 Safety

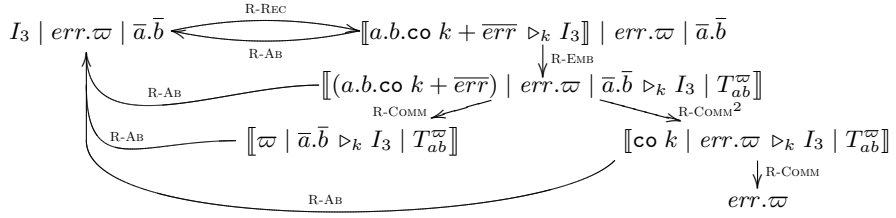
As discussion in the introduction, a safety test T^ϖ is a process that tests and reports "bad" behaviour on a channel ϖ; a process P *passes* the test if $P \mid T^\varpi$ cannot output on ϖ:

Definition 4 (Passing Safety Tests). *A process* P *passes a safety test* T^ϖ*, written* $P\,\text{cannot}\,T^\varpi$*, when* $P \mid T^\varpi \not\Downarrow_\varpi$.

Given a specification S, an implementation I preserves the liveness properties of S if every successful safety test of S is also a successful safety test of I. This leads us to the definition of *safety preservation*, which amounts to the inverse of the standard definition of *may-testing* [11].

Definition 5 (Safety Preservation). *I preserves the safety properties of S, and we write $S \sqsubseteq_{\text{safe}} I$, if for every safety test T^{ϖ} if S cannot T^{ϖ} then I cannot T^{ϖ}.*

Example 7. Consider the safety test $T^{\varpi}_{ab} = err.\varpi \mid \overline{a}.\overline{b}$. Transaction I_3 passes this test, because there is no possibility of reaching a top-level output on ϖ:



At no point do we have a top-level output on ϖ, so that I_3 passes this test. In fact, we have that $S_{ab} \sqsubseteq_{\text{safe}} I_3$ (we prove this formally in Ex. 12). I_4 however is not a safe implementation of S_{ab} because it does not pass this test:

$$I_4 \mid \overline{a}.\overline{b}.err.\varpi \xrightarrow{\text{R-Rec}} \xrightarrow{\text{R-Emb}} [\![(a.b.\mathtt{co}\ k \mid \overline{err}) \mid \overline{a}.\overline{b}.err.\varpi \triangleright_k I_4 \mid T^{\varpi}_{ab}]\!]$$
$$\xrightarrow{\text{R-Comm}^3} [\![\mathtt{co}\ k \mid \varpi \triangleright_k I_4 \mid T^{\varpi}_{ab}]\!] \xrightarrow{\text{R-Co}} \varpi \qquad \square$$

As we will prove by Thm. 3 in Sect. 6, liveness preservation implies safety preservation. Thus, to show that an implementation I preserves both the liveness and safety properties of a specification S, it suffices to show that $S \sqsubseteq_{\text{live}} I$.

5 Clean Traces and Safety

We give an overview of the definitions and results we reuse from previous work [22, 23]. These involve the definition of a Labelled Transition System (LTS) that describes the traces of processes, the definition of clean traces over this LTS, and a rephrasing of results about may-testing in terms of safety.

5.1 Labelled Transition System

The LTS (Fig(s). 3 and 4) is defined over an extension called TransCCS$^{\circ}$, ranged over by $\mathcal{P}, \mathcal{Q}$. Transactions in TransCCS$^{\circ}$ are distributed as a primary transaction, denoted by $[\![\mathcal{P} \triangleright_k \mathcal{Q}]\!]$, and zero or more secondary transactions, denoted by $[\![\mathcal{P} \triangleright_k \mathcal{Q}]\!]^{\circ}$ which correspond to embedded processes. This simulates embedding in the reduction semantics while keeping processes separate, supporting compositional reasoning. Internal actions in the LTS correspond to reduction steps up to this distribution of transactions; this is made precise in [22].

As an example, consider the trace starting with $I_3 \mid \overline{a}.\overline{b}.\omega$:

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

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

(eliding L-Trans for secondary transactions)

Fig. 3. LTS: Standard Actions

$$\begin{aligned}
&\mu X.[\![a.b.\mathtt{co}\ k + \overline{err}.\mathbf{0} \triangleright_k X]\!] \mid \overline{a}.\overline{b}.\omega \\
\xrightarrow{\tau}\xrightarrow{\mathtt{emb}\ k}\ &[\![a.b.\mathtt{co}\ k + \overline{err}.\mathbf{0} \triangleright_k I_3]\!] \mid [\![\overline{a}.\overline{b}.\omega \triangleright_k \overline{a}.\overline{b}.\omega]\!]^\circ \\
\xrightarrow{k(\tau)}\xrightarrow{k(\tau)}\ &[\![b.\mathtt{co}\ k \triangleright_k I_3]\!] \mid [\![\overline{b}.\omega \triangleright_k \overline{a}.\overline{b}.\omega]\!]^\circ \xrightarrow{\mathtt{co}\ k} \omega
\end{aligned}$$

Notice how (after unfolding the transaction once) the test is embedded and becomes a secondary transaction $[\![\overline{a}.\overline{b}.\omega \triangleright_k \overline{a}.\overline{b}.\omega]\!]^\circ$. Actions in the LTS are marked with the transactions that execute them (rule L-Trans); a primary and secondary k-transaction can communicate and therefore the action $k(a)$ of the primary k-transaction is matched by the action $k(\overline{a})$ of the secondary k-transaction (L-Comm), resulting in a $k(\tau)$ action.

Consider also the trace starting with $I_3 \mid T^{\varpi}_{ab}$ (*cf.* Ex. 7):

$$\begin{aligned}
&\mu X.[\![a.b.\mathtt{co}\ k + \overline{err}.\mathbf{0} \triangleright_k X]\!] \mid err.\varpi \mid \overline{a}.\overline{b} \\
\xrightarrow{\tau}\xrightarrow{\mathtt{emb}\ k}\ &[\![a.b.\mathtt{co}\ k + \overline{err}.\mathbf{0} \triangleright_k I_3]\!] \mid [\![err.\varpi \mid \overline{a}.\overline{b} \triangleright_k err.\varpi \mid \overline{a}.\overline{b}]\!]^\circ \\
\xrightarrow{k(\tau)}\ &[\![\mathbf{0} \triangleright_k I_3]\!] \mid [\![\varpi \mid \overline{a}.\overline{b} \triangleright_k err.\varpi \mid \overline{a}.\overline{b}]\!]^\circ
\end{aligned}$$

At this point, the transaction can only abort:

$$\xrightarrow{\mathtt{ab}\ k} \mu X.[\![a.b.\mathtt{co}\ k + \overline{err}.\mathbf{0} \triangleright_k X]\!] \mid err.\varpi \mid \overline{a}.\overline{b}$$

As in the reduction semantics, no trace of $I_3 \mid T^{\varpi}_{ab}$ leads to a top-level ϖ.

5.2 Clean Traces

There is an essential difference between the two traces of the previous section:

$$I_3 \mid T^{\omega}_{ab} \xrightarrow{\tau,\mathtt{emb}\ k,k(\tau),k(\tau),\mathtt{co}\ k} \omega \qquad\qquad I_3 \mid T^{\varpi}_{err} \xrightarrow{\tau,\mathtt{emb}\ k,k(\tau),\mathtt{ab}\ k} I_3 \mid T^{\varpi}_{err}$$

In the first, every action performed inside a transaction is eventually committed; in the second trace, however, the embedding step into the k transaction and the internal step within the k transaction are subsequently aborted and undone.

$$\text{B-CoPri}\ \frac{\mathcal{P} \equiv \mathcal{P}' \mid \mathtt{co}\ k}{[\![\mathcal{P} \triangleright_k \mathcal{Q}]\!] \xrightarrow{\mathtt{co}\ k} \mathcal{P}'} \qquad \text{B-CoSec}\ \frac{}{[\![\mathcal{P} \triangleright_k \mathcal{Q}]\!]^\circ \xrightarrow{\mathtt{co}\ k} \mathcal{P}} \qquad \text{B-Ab}\ \frac{}{[\![\mathcal{P} \triangleright_k \mathcal{Q}]\!] \xrightarrow{\mathtt{ab}\ k} \mathcal{Q}}$$

$$\text{B-Emb}\ \frac{}{\mathcal{P} \xrightarrow{\mathtt{emb}\ k} [\![\mathcal{P} \triangleright_k \langle\mathcal{P}\rangle]\!]^\circ} \qquad \text{B-Trans}\ \frac{\mathcal{P} \xrightarrow{\beta} \mathcal{P}' \qquad \beta \neq \mathtt{co}\ k, \mathtt{ab}\ k}{[\![\mathcal{P} \triangleright_k \mathcal{Q}]\!] \xrightarrow{\beta} [\![\mathcal{P}' \triangleright_k \mathcal{Q}]\!]} \qquad \text{B-Par}\ \frac{\mathcal{P} \xrightarrow{\beta} \mathcal{P}' \qquad \mathcal{Q} \xrightarrow{\beta} \mathcal{Q}'}{\mathcal{P} \mid \mathcal{Q} \xrightarrow{\beta} \mathcal{P}' \mid \mathcal{Q}'}$$

$$\text{B-Rec}\ \frac{}{\mu X.P \xrightarrow{\beta} \mu X.P} \qquad \text{B-Act}\ \frac{}{\sum \mu_i.P_i \xrightarrow{\beta} \sum \mu_i.P_i} \qquad \text{B-Co}\ \frac{}{\mathtt{co}\ k \xrightarrow{\beta} \mathtt{co}\ k} \qquad \text{B-Hide}\ \frac{\mathcal{P} \xrightarrow{\beta} \mathcal{P}'}{\nu a.\mathcal{P} \xrightarrow{\beta} \nu a.\mathcal{P}'}$$

(eliding B-Ab and B-Trans for secondary transactions)

Fig. 4. LTS: Broadcast actions

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

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

Fig. 5. Clean Traces

Clean traces are CCS traces that correspond to raw traces in the LTS where all transactions performing actions are eventually committed at the end of the trace. Formally, clean traces are specified by the relation $\mathcal{P} \xrightarrow{t}_\Delta \mathcal{P}'$, given in Fig. 5. The parameter Δ is used to record which transactions will commit, and hence which actions are allowed inside the trace.

Example 8. The clean trace $I_3 \mid T^\omega_{ab} \xrightarrow{\tau,\tau,\tau}_{\{k\}} \omega$ corresponds to the first trace above. In isolation, I_3 has the clean traces $I_3 \xrightarrow{\epsilon}_\emptyset I_3$ and $I_3 \xrightarrow{ab}_{\{k\}} \mathbf{0}$, but not the singleton trace a: we need $k \in \Delta$ to do the a action inside the transaction, but we cannot derive $I_3 \xrightarrow{a}_{\{k\}}$ since the transaction cannot yet commit having done only the a action. Clean traces are hence not prefix closed. □

Usually, we care only that there is *some* Δ for which $\mathcal{P} \xrightarrow{t}_\Delta \mathcal{P}'$ can be derived, which motivates the following definition:

Definition 6. *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 $\mathcal{P} \stackrel{t}{\Rightarrow}_{CL}$ to denote that t is a weak clean trace of $\mathcal{P}$.*

5.3 Characterization of Safety as Clean Trace Inclusion

Safety preservation is characterized by clean trace inclusion [22].

Definition 7 (Language). *The language of a process $\mathcal{P}$ is the set of weak clean traces it can do:*

$$\mathcal{L}(\mathcal{P}) \stackrel{\text{def}}{=} \{t \mid \mathcal{P} \stackrel{t}{\Rightarrow}_{CL}\}$$

Theorem 1 (Safety preservation). $P \sqsubseteq_{\text{safe}} Q$ *iff* $\mathcal{L}(P) \supseteq \mathcal{L}(Q)$.

6 Characterization of Liveness as Clean Tree Failures

We now proceed with the main technical result of this paper: a sound and complete characterization of liveness preservation in terms of *clean tree failures*. In this section we present the model, give a number of examples, and state the main results. The proof of soundness and completeness is summarized in Sect. 8.

The intuition of our model is that $\mathcal{P}$ has a clean tree failure (t, Ref) iff $\mathcal{P}$ can do a clean trace to $\mathcal{P}'$ and $\mathcal{P}'$ cannot do any of the clean traces in the set Ref.

Definition 8 (Tree failures). *Tree failures are defined as*

$$\mathcal{F}(\mathcal{P}) \stackrel{\text{def}}{=} \{(t, \mathit{Ref}) \mid \exists \mathcal{P}'.\ \mathcal{P} \stackrel{t}{\Rightarrow}_{CL} \mathcal{P}' \text{ and } \mathcal{L}(\mathcal{P}') \cap \mathit{Ref} = \emptyset\}$$

Theorem 2 (Liveness Preservation). $P \sqsubseteq_{\text{live}} Q$ *iff* $\mathcal{F}(P) \supseteq \mathcal{F}(Q)$.

Example 9. Consider the transactions S_{ab} and I_3 from Fig. 1. The only clean traces either of these processes can do is the empty trace ϵ and the trace ab; moreover, for either process, the only clean trace that they cannot refuse after the empty trace is the trace ab, and both can refuse all clean traces after the trace ab. Hence, the set of failures for both processes is

$$\{(\epsilon, \mathit{Ref}) \mid ab \notin \mathit{Ref}\} \cup \{(ab, \mathit{Ref}) \mid \text{any } \mathit{Ref}\}$$

so that by Thm. 2 we have $S_{ab} \simeq_{\text{live}} I_3$. □

Our model is simpler than the model of liveness preservation in CCS [21]. This is due to the existence of transactional tests that do not allow processes to deadlock while they communicate with these tests, as shown in Ex. 5.

As in CCS, liveness preservation implies safety preservation.

Theorem 3 (Liveness implies safety). *If* $P \sqsubseteq_{\text{live}} Q$ *then* $P \sqsubseteq_{\text{safe}} Q$.

Proof. By Thm. 1 it suffices to prove that if t is a clean trace of Q then it is a clean trace of P. Let t be a clean trace of Q; then $(t, \emptyset) \in \mathcal{F}(Q)$ and by Thm. 2, $(t, \emptyset) \in \mathcal{F}(P)$. Thus t is a clean trace of P. □

7 Canonical Tests

We identify a class of canonical liveness tests that encode sufficient power to distinguish any processes P and Q for which $P \not\sqsubseteq_{\text{live}} Q$. We use these tests in the definition of a restricted form of liveness preservation, which we will show by Prop. 8 in the following section implies inverse failure inclusion. This result is crucial to show completeness of our characterization, but also implies that restricted liveness coincides with standard liveness.

Definition 9 (T^{ω}_{Ref}). *If Ref is a set of clean traces, we define the liveness test*

$$T^{\omega}_{Ref} \stackrel{\text{def}}{=} \mu X.\, \left[\!\left[\textstyle\sum_{t \in Ref} \tau.\overline{t}.(\mathtt{co}\ k \mid \omega) \rhd_k X \right]\!\right]$$

Definition 10 ($T^{\omega}_{(t,Ref)}$). *If t is a clean trace and Ref a set of clean traces, we define the liveness test $T^{\omega}_{(t,Ref)}$ by induction on t:*

$$T^{\omega}_{(\epsilon,Ref)} \stackrel{\text{def}}{=} T^{\omega}_{Ref} \qquad T^{\omega}_{(at,Ref)} \stackrel{\text{def}}{=} \omega + \overline{a}.T^{\omega}_{(t,Ref)}$$

These tests are interesting because (as we will show in Sect. 8) a process P passes the liveness test $T^{\omega}_{(t,Ref)}$ exactly if (t, Ref) is not a failure of P. Note that P fails the liveness test $T^{\omega}_{(t,Ref)}$ only if it can do a clean trace t and then refuse to do all the traces of Ref.

Example 10. The liveness test T^{ω} we considered in Ex. 5 is exactly the test $T^{\omega}_{(a,\{bc\})}$. We saw that $P = a.(b.c + b.d)$ passes this test, but $Q = a.b.c + a.b.d$ does not. Hence, $(a, \{bc\})$ is a failure of Q but not of P. □

Definition 11 (Restricted Liveness Preservation ($\hat{\sqsubseteq}_{\mathbf{live}}$)).

$$P \,\hat{\sqsubseteq}_{\text{live}}\, Q \quad \stackrel{\text{def}}{=} \quad \forall t, Ref.\ \ \textit{if}\ P \,\text{shd}\, T^{\omega}_{(t,Ref)}\ \textit{then}\ Q \,\text{shd}\, T^{\omega}_{(t,Ref)}$$

Theorem 4. $(\sqsubseteq_{\text{live}}) = (\hat{\sqsubseteq}_{\text{live}})$.

Proof. Follows by Prop. 6 (soundness) and Prop. 8 in the following section. □

8 Soundness and Completeness

We now outline the proof that the characterization of the fair-testing preorder in terms of clean tree failures is sound and complete. This proof makes use of the ability to zip and unzip clean traces, proved in [22, 23], which means that processes can communicate independently of their transaction structure.

Proposition 4 (Clean unzipping). *If $P \mid Q \stackrel{\epsilon}{\Rightarrow}_{CL} \mathcal{R}$ then there exist t, $\mathcal{P}'$, and $\mathcal{Q}'$ such that $P \stackrel{t}{\Rightarrow}_{CL} \mathcal{P}'$ and $Q \stackrel{\overline{t}}{\Rightarrow}_{CL} \mathcal{Q}'$ and $\mathcal{R}$ is equal up to merging of distributed transactions with $\mathcal{P}' \mid \mathcal{Q}'$.*

Proposition 5 (Clean zipping). *If $P \stackrel{t}{\Rightarrow}_{CL} \mathcal{P}'$ and $Q \stackrel{\overline{t}}{\Rightarrow}_{CL} \mathcal{Q}'$ then there exists an $\mathcal{R}$ such that $P \mid Q \stackrel{\epsilon}{\Rightarrow}_{CL} \mathcal{R}$ and $\mathcal{R}$ is equal up to merging of distributed transactions with $\mathcal{P}' \mid \mathcal{Q}'$.*

The following theorem is key in the proof of soundness and completeness, and states that we can construct clean traces from raw traces:

Theorem 5 (Clean trace construction). *If $\mathcal{P} \stackrel{\epsilon}{\Rightarrow} \mathcal{R}$ then there exists $\mathcal{R}'$ such that $\mathcal{P} \stackrel{\epsilon}{\Rightarrow}_{CL} \mathcal{R}'$ and*

1. *If* $\mathcal{R}\downarrow_\omega$ *then* $\mathcal{R}'\downarrow_\omega$ *(success is preserved)*
2. *If* $\mathcal{R}\Downarrow_\omega$ *then* $\mathcal{R}'\Downarrow_\omega$ *(failure is preserved)*

This theorem strengthens an earlier result [22], where we proved (1) but not (2); the proof is however significantly different. Intuitively, the construction of the clean trace postpones all commits to the end of the trace and aborts all actions that are never committed in the raw trace.

Definition 12. $\mathcal{L}^\omega(\mathcal{P}) \stackrel{\text{def}}{=} \{t \mid \mathcal{P} \stackrel{t}{\Rightarrow}_{CL}, \omega \notin t\}$

The proof of soundness is based on the construction of a clean trace from a raw trace, and zipping and unzipping of clean traces.

Proposition 6 (Soundness). *If* $\mathcal{F}(P) \supseteq \mathcal{F}(Q)$ *then* $P \sqsubseteq_{\text{live}} Q$.

Proof. Assume $\mathcal{F}(P) \supseteq \mathcal{F}(Q)$. We prove the contrapositive of $P \sqsubseteq_{\text{live}} Q$: suppose $\neg(Q \operatorname{shd} T)$ for some test T; we have to show that $\neg(P \operatorname{shd} T)$.

Since $\neg(Q \operatorname{shd} T)$, there exists an $\mathcal{R}$ such that $Q \mid T \stackrel{\epsilon}{\Rightarrow} \mathcal{R}\Downarrow_\omega$. Hence by Thm. 5, there exists $\mathcal{R}'$ such that $Q \mid T \stackrel{\epsilon}{\Rightarrow}_{\text{CL}} \mathcal{R}'\Downarrow_\omega$. By Prop. 4, there exist $t, \mathcal{Q}', \mathcal{T}'$ such that $Q \stackrel{t}{\Rightarrow}_{\text{CL}} \mathcal{Q}'$ and $T \stackrel{\bar{t}}{\Rightarrow}_{\text{CL}} \mathcal{T}'$, where $\mathcal{R}'$ is equal to $\mathcal{Q}' \mid \mathcal{T}'$ up to distribution of transactions. Define $Ref = \{t' \mid \mathcal{T}' \stackrel{\bar{t}'}{\Rightarrow}_{\text{CL}} \mathcal{T}'' \mid \omega\}$. Then $\mathcal{L}(\mathcal{Q}') \cap Ref$ must be empty, because otherwise $\mathcal{R}'\Downarrow_\omega$ by zipping the clean traces. Hence $(t, Ref) \in \mathcal{F}(\mathcal{Q})$ and therefore $(t, Ref) \in \mathcal{F}(\mathcal{P})$. It follows that exists $\mathcal{P}'$ such that $P \stackrel{t}{\Rightarrow}_{\text{CL}} \mathcal{P}'$ and $\mathcal{L}(\mathcal{P}') \cap Ref = \emptyset$. By Prop. 5, $P \mid T \stackrel{t'}{\Rightarrow}_{\text{CL}} \mathcal{P}' \mid \mathcal{T}'$ where $\mathcal{P}' \mid \mathcal{T}'\Downarrow_\omega$ since $\mathcal{L}(\mathcal{P}') \cap Ref = \emptyset$. Therefore $\neg(P \operatorname{shd} T)$. □

The proof of completeness makes essential use of the canonical tests (Sect. 7).

Lemma 1. $\mathcal{L}^\omega(\mathcal{T}_{Ref}) = \overline{Ref}$

Proof. By definition of $\mathcal{L}^\omega()$ and $\mathcal{T}_{Ref}$. Note that $\mathcal{L}^\omega(\mathcal{T}_{Ref})$ is not the prefix closure of Ref because we used a transactional $\mathcal{T}_{Ref}$. □

Lemma 2. $\mathcal{L}(\mathcal{P}) \cap Ref = \emptyset$ *iff* $(\mathcal{P} \mid \mathcal{T}_{Ref})\Downarrow_\omega$.

Proof. (Only if) By contradiction: Assume $(\mathcal{P} \mid \mathcal{T}_{Ref})\Downarrow_\omega$, i.e. $\mathcal{P} \mid \mathcal{T}_{Ref} \stackrel{\epsilon}{\Rightarrow} \mathcal{R}\downarrow_\omega$. Then by Thm. 5 there exists $\mathcal{R}'$ such that $\mathcal{P} \mid \mathcal{T}_{Ref} \stackrel{\epsilon}{\Rightarrow}_{\text{CL}} \mathcal{R}'\downarrow_\omega$. We can therefore apply Prop. 4 to get a clean trace $t \in \mathcal{L}(\mathcal{P})$ and $\bar{t} \in \mathcal{L}^\omega(\mathcal{T}_{Ref})$. Hence by Lem. 1 we must have $t \in Ref$, contradicting the assumption that $\mathcal{L}(\mathcal{P}) \cap Ref = \emptyset$.

(If) By contradiction: Assume there exists $t \in \mathcal{L}(\mathcal{P}) \cap Ref$. Then we can apply Prop. 5 to get $\mathcal{P} \mid \mathcal{T}_{Ref} \stackrel{\epsilon}{\Rightarrow}_{\text{CL}} \mathcal{P}'\downarrow_\omega$ contradicting $(\mathcal{P} \mid \mathcal{T}_{Ref})\Downarrow_\omega$. □

Lemma 3. $\mathcal{L}^\omega(\mathcal{T}_{(t,Ref)}) = \{\bar{t}_1 \mid \exists t_2.\ t = t_1 t_2\} \cup \{\bar{t}t' \mid t' \in \mathcal{L}^\omega(\mathcal{T}_{Ref})\}$

Proof. By definition of $\mathcal{L}^\omega()$ and $\mathcal{T}_{(t,Ref)}$. □

A process fails a canonical liveness test iff it has the corresponding tree failure.

Proposition 7 (Tests and Failures). $(t, Ref) \in \mathcal{F}(P)$ *iff* $\neg(P \operatorname{shd} T_{(t,Ref)})$.

Proof. (Only if) Let $(t, Ref) \in \mathcal{F}(P)$, i.e. $\exists \mathcal{P}'.\ P \stackrel{t}{\Rightarrow}_{\text{CL}} \mathcal{P}'$ and $\mathcal{L}(\mathcal{P}') \cap Ref = \emptyset$. Clearly $T_{(t,Ref)} \stackrel{\bar{t}}{\Rightarrow}_{\text{CL}} T_{Ref}$ so that, by Prop. 5, $P \mid T_{(t,Ref)} \stackrel{\epsilon}{\Rightarrow}_{\text{CL}} \mathcal{R}$ equal up to merging of transactions to $\mathcal{P}' \mid T_{Ref}$. It follows from Lem. 2 that success is not reachable from this state.

(If) Since $\neg(P \operatorname{shd} T_{(t,Ref)})$ it follows that $\exists \mathcal{R}.\ P \mid T_{(t,Ref)} \stackrel{\epsilon}{\Rightarrow} \mathcal{R}\Downarrow_{\omega}$. Then by Thm. 5, $\exists \mathcal{R}'.\ P \mid T_{(t,Ref)} \stackrel{\epsilon}{\Rightarrow}_{\text{CL}} \mathcal{R}'\Downarrow_{\omega}$. By Prop. 4, there exist $t', \mathcal{P}', \mathcal{T}'$ such that $P \stackrel{t'}{\Rightarrow}_{\text{CL}} \mathcal{P}'$ and $T_{(t,Ref)} \stackrel{\bar{t}'}{\Rightarrow}_{\text{CL}} \mathcal{T}'$, and $\mathcal{P}' \mid \mathcal{T}'\Downarrow_{\omega}$. Thus, $t' \in \mathcal{L}(P)$ and by Lem. 3

$$\bar{t}' \in \mathcal{L}^{\omega}(T_{(t,Ref)}) = \{\bar{t}_1 \mid \exists t_2.\ t = t_1\, t_2\} \cup \{\bar{t}\, t_2 \mid t_2 \in \mathcal{L}^{\omega}(\mathcal{T}_{Ref})\}$$

We take cases on $\bar{t}' \in \mathcal{L}^{\omega}(\mathcal{T}_{(t,Ref)})$:

1. $\bar{t}' = \bar{t}t_2$ for some $t_2 \in \mathcal{L}^{\omega}(T_{Ref})$. Not possible, because then $\mathcal{T}' = \omega\downarrow_{\omega}$.
2. $\bar{t}' = \overline{t_1}$ for some $t_1 t_2 = t$ with t_2 non-empty; again, not possible because then $\mathcal{T}' = \omega + \mathcal{T}''\downarrow_{\omega}$.
3. $\bar{t}' = \bar{t}$. Then $\mathcal{T}' = T_{Ref}$ and by Lem. 2 $\mathcal{L}(\mathcal{P}') \cap Ref = \emptyset$. Hence $(t, Ref) \in \mathcal{F}(\mathcal{P})$. □

Restricted liveness preservation implies inverse failure inclusion.

Proposition 8. *If $P \hat{\sqsubseteq}_{\text{live}} Q$ then $\mathcal{F}(P) \supseteq \mathcal{F}(Q)$.*

Proof. Let $(t, Ref) \in \mathcal{F}(Q)$. By Prop. 7 we have $\neg(Q \operatorname{shd} T_{(t,Ref)})$, therefore $\neg(P \operatorname{shd} T_{(t,Ref)})$ since $P \sqsubseteq_{\text{live}} Q$, and finally $(t, Ref) \in \mathcal{F}(P)$ by Prop. 7. □

Corollary 1 (Completeness). *If $P \sqsubseteq_{\text{live}} Q$ then $\mathcal{F}(P) \supseteq \mathcal{F}(Q)$.*

Proof. By the definitions of ($\sqsubseteq_{\text{live}}$) and ($\hat{\sqsubseteq}_{\text{live}}$) and Prop. 8. □

9 Weak Clean-Trace Bisimilarity

In this section we present a convenient coinductive proof technique for liveness preservation, which is based on weak bisimilarity over clean traces. We show that this technique is sound but not complete with respect to liveness preservation, and use it to prove liveness and safety preservation.

Definition 13 (Weak Clean-Trace Bisimulation). *Θ is a weak clean-trace bisimulation if whenever $(\mathcal{P}, \mathcal{Q}) \in \Theta$ the following two conditions are satisfied.*

1. *$\forall t, \mathcal{P}'.\ \mathcal{P} \stackrel{t}{\Rightarrow}_{CL} \mathcal{P}'$ we have $\exists \mathcal{Q}'$ such that $\mathcal{Q} \stackrel{t}{\Rightarrow}_{CL} \mathcal{Q}'$ and $(\mathcal{P}', \mathcal{Q}') \in \Theta$,*
2. *$\forall t, \mathcal{Q}'.\ \mathcal{Q} \stackrel{t}{\Rightarrow}_{CL} \mathcal{Q}'$ we have $\exists \mathcal{P}'$ such that $\mathcal{P} \stackrel{t}{\Rightarrow}_{CL} \mathcal{P}'$ and $(\mathcal{P}', \mathcal{Q}') \in \Theta$.*

Weak clean-trace bisimilarity, denoted by $\approx$, is the largest weak clean-trace bisimulation.

Weak clean-trace bisimilarity is sound with respect to both liveness and safety.

Theorem 6 (Soundness of $(\approx)$). *If $\mathcal{P} \approx \mathcal{Q}$ then $\mathcal{P} \eqsim_{\text{live}} \mathcal{Q}$ and $\mathcal{P} \eqsim_{\text{safe}} \mathcal{Q}$.*

Proof. Since $\approx$ is commutative, it suffices to prove that $\mathcal{P} \sqsubseteq_{\text{live}} \mathcal{Q}$ and hence by Thm. 2 that $\mathcal{F}(\mathcal{P}) \supseteq \mathcal{F}(\mathcal{Q})$. Let $(t, S) \in \mathcal{F}(\mathcal{Q})$, i.e. $\mathcal{P} \stackrel{t}{\Rightarrow}_{\text{CL}} \mathcal{P}'$ where $\mathcal{L}(\mathcal{P}') \cap S = \emptyset$. Since $\mathcal{P} \approx \mathcal{Q}$, $\exists \mathcal{Q}'$ such that $\mathcal{Q} \stackrel{t}{\Rightarrow}_{\text{CL}} \mathcal{Q}'$ where $\mathcal{P}' \approx \mathcal{Q}'$. It remains to show that $\mathcal{L}(\mathcal{Q}') \cap S = \emptyset$. We proceed by contradiction. Suppose that $\exists t' \in \mathcal{L}(\mathcal{Q}') \cap S$. Then $\exists \mathcal{Q}''$ such that $\mathcal{Q}' \stackrel{t'}{\Rightarrow}_{\text{CL}} \mathcal{Q}''$. But then since $\mathcal{P}' \approx \mathcal{Q}'$, $\exists \mathcal{P}''$ such that $\mathcal{P}' \stackrel{t'}{\Rightarrow}_{\text{CL}} \mathcal{P}''$, contradicting the assumption that $\mathcal{L}(\mathcal{P}') \cap S = \emptyset$. □

The following example shows that weak clean-trace bisimilarity is not complete with respect to liveness and safety.

Example 11. Consider the processes $P = a.(\tau.b + \tau.c)$ and $Q = a.b + a.c$. The two processes are not bisimilar: P can do a clean trace $P \stackrel{a}{\Rightarrow}_{\text{CL}} (\tau.b + \tau.c)$ and Q can only follow it by $Q \stackrel{a}{\Rightarrow}_{\text{CL}} b$ or $Q \stackrel{a}{\Rightarrow}_{\text{CL}} c$. Neither b nor c are bisimilar to $(\tau.b + \tau.c)$. It is not difficult to show, however, that $P \eqsim_{\text{live}} Q$ (and thus, by Thm. 3, $P \eqsim_{\text{may}} Q$) by observing that any tree failure $(t, S) \in \mathcal{F}(P)$ is a tree failure of Q and vice versa. □

The next result simplifies reasoning about weak clean-trace bisimilarity by allowing us to consider a single unfolding of recursive transactions.

Proposition 9. *If $[\![P \triangleright_k \mathtt{0}]\!] \approx [\![Q \triangleright_k \mathtt{0}]\!]$ then $\mu X.[\![P \triangleright_k X]\!] \approx \mu X.[\![Q \triangleright_k X]\!]$.*

Proof. By enumeration of the clean traces of the restarting transactions, which start with a number of aborts and continue with a clean trace of the non-restarting transactions. □

We use weak clean-trace bisimilarity to give simple coinductive proofs of liveness and safety preservation for the examples of the introduction and Fig. 1.

Example 12. Recall once more transactions S_{ab} and I_3 (Fig. 1). We prove that $S_{ab} \eqsim_{\text{live}} I_3$ and $S_{ab} \eqsim_{\text{may}} I_3$ by showing $S_{ab} \approx I_3$. By Prop. 9 it suffices to show $[\![a.b.\mathtt{co}\ k \triangleright_k \mathtt{0}]\!] \approx [\![a.b.\mathtt{co}\ k + \overline{\overline{err}}.\mathtt{0} \triangleright_k \mathtt{0}]\!]$, which can be easily proved by showing that the relation containing the two transactions and $(\mathtt{0}, \mathtt{0})$ is a bisimulation. □

Example 13. We now turn our attention to the example of the introduction. We show that the implementation *Sys* preserves both the liveness and safety properties of the specification *Spec*. In fact, we prove the stronger results that *Spec* $\eqsim_{\text{live}}$ *Sys* and *Spec* $\eqsim_{\text{safe}}$ *Sys* by showing that *Spec* $\approx$ *Sys*. Consider the relation $\Theta = \{(Spec, Sys)\} \cup \{ (\overline{rec}\langle v\rangle,\ \nu q.\nu s.\ (Trg_{del,s,q} \mid \overline{q}\langle v\rangle \mid \overline{s}\langle 1\rangle)) \mid \exists v\} \cup \{(\mathtt{0}, \mathtt{0})\}$. It is easy to verify that Θ is a weak clean-trace bisimulation. □

10 Related work

The study of safety and liveness in concurrent languages goes back thirty years [18, 19], but although there are many studies of (isolated) transactions in concurrent languages [1–6, 9, 12, 15, 17] none of them study liveness.

There is much less research on *communicating* transactions. We are aware of only three other studies: Committed π [7], RCCS [10], and Transactors [13], none of which discuss safety or liveness properties of transactions.

Most closely related is the Committed π calculus where, like in TransCCS, transactions must be combined before they can communicate. However, they are merged rather than embedded: $[\![P_1 \triangleright Q_1]\!] \mid [\![P_2 \triangleright Q_2]\!] \to [\![P_1 \mid P_2 \triangleright Q_1 \mid Q_2]\!]$. This leads to pessimistic rollback behaviour: when transactions communicate and a failure happens, *all* transactions must be rolled back to their initial state. Moreover, Committed π includes an explicit abort construct, which makes uncommitted actions observable [22]. For example, the transaction $[\![a.\mathbf{0} \triangleright \mathbf{0}]\!]$ can be distinguished from $\mathbf{0}$ by $[\![\overline{a}.\mathtt{ab} \triangleright \omega]\!]$.

Reversible CCS extends CCS with reversible actions which can be rolled back and irreversible actions which act as a commit. The most important difference with TransCCS is that in RCCS a commit by a single transaction will cause the commit of all transactions it has communicated with. For example, the RCCS transaction $[\![a.\mathbf{0} \triangleright_k \mathbf{0}]\!]$ can be distinguished from $\mathbf{0}$ by $[\![\overline{a}.(\mathtt{co}\ l \mid \omega) \triangleright_l \mathbf{0}]\!]$.

Finally, Transactors is an extension of the actors model with communicating transactions. It is a much lower level language than TransCCS with a more complicated semantics, but it is similar in intent: for instance, $[\![a.\mathbf{0} \triangleright_k \mathbf{0}]\!]$ seems indistinguishable from $\mathbf{0}$, although in the absence of a behavioural theory for the language this is difficult to show.

We studied liveness properties of communicating transactions under an assumption of fairness, which must be guaranteed by potential implementations of the language. There is some work that investigates the fairness guarantees that can be offered by implementations of isolated transactions [16, 20]; an extension of those studies to communicating transactions would be worthwhile.

11 Conclusions

We studied liveness and safety in TransCCS; to the extent of our knowledge, this is the first semantic study of liveness in the presence of transactions. We showed that fair-testing and may-testing capture the right notions of liveness and safety and gave numerous examples to build useful intuitions. We developed a sound and complete characterization of liveness preservation in terms of clean tree failures, extending our earlier work on clean traces. This characterization is simpler than the characterization of liveness preservation in CCS, made possible by the additional distinguishing power added by transactions. We also gave a coinductive proof technique for liveness preservation based on weak clean trace bisimulation, which we proved to be sound but incomplete. We used the characterization and the bisimulation in example proofs of liveness preservation.

Further study of weak bisimulation and other proof techniques is future work. For instance, it is unclear at present whether bisimilarity preserves all contexts and what its characterization is. We also plan to extend TransCCS to the π-calculus. Finally, we intend 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. Black, A.P., Cremet, V., Guerraoui, R., Odersky, M.: An equational theory for transactions. In: FSTTCS. LNCS, vol. 2914, pp. 38–49. Springer (2003)
3. Bocchi, L.: Compositional nested long running transactions. In: FASE. LNCS, vol. 2984, pp. 194–208. Springer (2004)
4. Bruni, R., Laneve, C., Montanari, U.: Orchestrating transactions in join calculus*. In: CONCUR. LNCS, vol. 2421, pp. 531–544. Springer (2002)
5. Bruni, R., Melgratti, H., Montanari, U.: Theoretical foundations for compensations in flow composition languages. In: POPL. pp. 209–220. ACM (2005)
6. Bruni, R., Melgratti, H., Montanari, U.: Nested commits for mobile calculi: extending Join. In: IFIP-TCS. pp. 569–582. Kluwer Academic Publishers (2004)
7. Buscemi, M.G., Melgratti, H.: Transactional service level agreement. In: TGC. pp. 124–139 (2008)
8. Cacciagrano, D., Corradini, F., Palamidessi, C.: Explicit fairness in testing semantics. Logical Methods in Computer Science 5(2) (2009)
9. Caires, L., Ferreira, C., Vieira, H.T.: A process calculus analysis of compensations. In: TGC. LNCS, vol. 5474, pp. 87–103 (2008)
10. Danos, V., Krivine, J.: Transactions in RCCS. In: CONCUR. LNCS, vol. 3653, pp. 398–412. Springer-Verlag (2005)
11. De Nicola, R., Hennessy, M.C.B.: Testing equivalences for processes. Theoretical Computer Science 34(1–2), 83–133 (1984)
12. Elmas, T., Qadeer, S., Tasiran, S.: A calculus of atomic actions. In: POPL. pp. 2–15 (2009)
13. Field, J., Varela, C.A.: Transactors: a programming model for maintaining globally consistent distributed state in unreliable environments. In: POPL. pp. 195–208 (2005)
14. van Glabbeek, R.: The linear time–branching time spectrum after 20 years (2009), Celebration of 20 years of CONCUR
15. Gorrieri, R., Marchetti, S., Montanari, U.: A^2CCS: atomic actions for CCS. Theor. Comp. Sci. 72(2-3), 203–223 (1990)
16. Guerraoui, R., Kapalka, M.: How Live Can a Transactional Memory Be? Tech. rep., EPFL (2009)
17. Harris, T., Marlow, S., Peyton-Jones, S., Herlihy, M.: Composable memory transactions. In: PPoPP. pp. 48–60. ACM (2005)
18. Lamport, L.: Proving the correctness of multiprocess programs. IEEE Trans. Softw. Eng. 3(2), 125–143 (1977)
19. Owicki, S., Lamport, L.: Proving liveness properties of concurrent programs. ACM Trans. Program. Lang. Syst. 4(3), 455–495 (1982)
20. Pedone, F., Guerraoui, R.: On transaction liveness in replicated databases. In: PRFTS. p. 104 (1997)
21. Rensink, A., Vogler, W.: Fair testing. Inf. and Comp. 205(2), 125–198 (2007)
22. de Vries, E., Koutavas, V., Hennessy, M.: Communicating Transactions. In: CONCUR (2010), *To appear*
23. de Vries, E., Koutavas, V., Hennessy, M.: Communicating transactions—technical appendix (April 2010), available at `http://www.scss.tcd.ie/Edsko.de.Vries`