A theory of communicating transactions

Matthew Hennessy

joint work with Edsko de Vries, Vasileois Koutavas

CHoCoLa, PPS Paris December 2012

Outline

Introduction

TransCCS

Liveness and safety properties

Compositional semantics
Standard Transactions

- Transactions provide *an abstraction for error recovery* in a concurrent setting.
- **Guarantees:**
  - **Atomicity:** Each transaction either runs in its entirety (commits) or not at all
  - **Consistency:** When faults are detected the transaction is automatically rolled-back
  - **Isolation:** The effects of a transaction are concealed from the rest of the system until the transaction commits
  - **Durability:** After a transaction commits, its effects are permanent
- **Isolation:**
  - good: provides coherent semantics
  - bad: limits concurrency
  - bad: limits co-operation between transactions and their environments

Communicating/Co-operating Transactions

- We *drop isolation to increase concurrency*
  - There is no limit on the communication between a transaction and its environment
- These new transactional systems guarantee:
  - **Atomicity:** Each transaction will either run in its entirety or not at all
  - **Consistency:** When faults are detected the transaction is automatically rolled-back, *together with all effects of the transaction on its environment*
  - **Durability:** After *all transactions that have interacted* commit, their effects are permanent (coordinated checkpointing)
**Example: three-way rendezvous**

\[ P_1 \parallel P_2 \parallel P_3 \parallel P_4 \]

**Problem:**

- \( P_n \) process/transaction subject to failure
- Some three \( P_n \) should decide to collaborate

**Result:**

- Each \( P_n \) in the coalition outputs id of its partners on channel \( out_n \)

**Example: programming a three-way rendezvous**

\[ P_1 \parallel P_2 \parallel P_3 \parallel P_4 \]

**Algorithm for \( P_n \):**

- Broadcast id \( n \) randomly to two arbitrary partners
  \( b!(n) \mid b!(n) \)
- Receive ids from two random partners
  \( b?(y) . b?(z) \)
- Propose coalition with these partners
  \( s_y!(n,z) . s_z!(n,y) \)
- Confirm that partners are in agreement:
  - if YES, commit and report
  - if NO, abort&retry
Example: programming a three-way rendezvous

\[ P_1 \parallel P_2 \parallel P_3 \parallel P_4 \]

\[ P_n \leftarrow b!(n) | b!(n) | \]

\[ \text{atomic} [ b?(y) . b?(z) . s_y!(n,z) . s_z!(n,y) . \text{proposing} ] \]

\[ s_n?(y_1, z_1) . s_n?(y_2, z_2) . \text{confirming} \]

\[ \text{if } \{y, z\} = \{y_1, z_1\} = \{y_2, z_2\} \]

\[ \text{then } \text{commit} \mid \text{out}_n!(y,z) \]

\[ \text{else } \text{abrt}&\text{retry} \]

Communicating Transactions: Issues

- **Language Design**
  - Transaction Synchronisers (Luchangco et al 2005)
  - Transactional Events for ML (Fluet, Grossman et al. ICFP 2008)
  - Communication Memory Transactions (Lesani, Palsberg PPoPP 2011)
  - ...

- **Implementation strategies**
  - See above

- **Semantics** Behavioural theory: what should happen when programs are run
  - TransCCS (Concur 2010, Aplas 2010)
Communication Memory Transactions  Lesani Palsberg

- Builds on optimistic semantics of memory transactions  O’Herlihy et al 2010
- Adds asynchronous channel-based message passing  as in Actors CML etc
- Formal reduction semantics
- Formal properties of semantics proved
- Implementation as a Scala library
- Performance evaluation using benchmarks

TransCCS

An extension of CCS with communicating transactions.

1. Simple language: 2 additional language constructs and 3 additional reduction rules.
2. Intricate concurrent and transactional behaviour:
   - encodes nested, restarting, and non-restarting transactions
   - does not limit communication between transactions
3. Simple behavioural theory: based on properties of systems:
   - Safety property: nothing bad happens
   - Liveness property: something good happens
TransCCS

Syntax: \( P, Q ::= \sum \mu_i . P_i \) guarded choice
\( P \parallel Q \) parallel
\( \nu a . P \) hiding
\( \mu X . P \) recursion
\( [P \triangleright_k Q] \) transaction \((k\text{ bound in } P)\)
\( \text{co } k \) commit

Transaction \([P \triangleright_k Q]\)

- execute \( P \) to completion \((\text{co } k)\)
- subject to random aborts
- if aborted roll back all effects of \( P \) and initiate \( Q \)
- roll back includes \ldots \text{environmental impact of } P

Rollbacks and Commits

Co-operating actions: \( a \leftarrow \text{needs co-operation of } \to \bar{a} \)

\[ T_a \parallel T_b \parallel T_c \parallel P_d \parallel P_e \]

where
\[
\begin{align*}
T_a &= [\bar{d} . b . (\text{co } k_1 \mid a) \triangleright_{k_1} 0] \\
T_b &= [\bar{c} . (\text{co } k_2 \mid b) \triangleright_{k_2} 0] \\
T_c &= [\bar{e} . c . \text{co } k_3 \triangleright_{k_3} 0] \\
P_d &= d . R_d \\
P_e &= e . R_e
\end{align*}
\]

- if \( T_c \) aborts, what roll-backs are necessary?
- When can action \( a \) be considered permanent?
- When can code \( R_d \) be considered permanent?
Reduction semantics main rules

R-COMM

\[ a_i = \overline{b_j} \]

\[ \sum_{i \in I} a_i \cdot P_i \mid \sum_{j \in J} b_j \cdot Q_j \rightarrow P_i \mid Q_j \]

Communication

R-CO

\[ \begin{bmatrix} P \mid k \triangleright_k Q \end{bmatrix} \rightarrow P \]

Commit

R-AB

\[ \begin{bmatrix} P \triangleright_k Q \end{bmatrix} \rightarrow Q \]

Random abort

R-EMB

\[ \begin{bmatrix} P \triangleright_k Q \end{bmatrix} \mid R \rightarrow \begin{bmatrix} P \mid R \triangleright_k Q \mid R \end{bmatrix} \]

Embed

Simple Example

Convention:

- ω: I am happy
- ω: I am sad

\[ a.c.ω + e.ω \mid \begin{bmatrix} \overline{\overline{a.c} . co} k + \overline{e} \triangleright_k r \end{bmatrix} \]

R-EMB

\[ \begin{bmatrix} a.c.ω + e.ω \mid \overline{\overline{\overline{a.c} . co}} k + \overline{e} \triangleright_k a.c.ω + e.ω \mid r \end{bmatrix} \]

R-COMM

\[ \begin{bmatrix} c.ω \mid \overline{\overline{c} . co} k \triangleright_k a.c.ω + e.ω \mid r \end{bmatrix} \]

R-COMM

\[ \begin{bmatrix} ω \mid co k \triangleright_k a.c.ω + e.ω \mid r \end{bmatrix} \]

R-CO

ω
Simple Example (a second trace)

\[ a.c.\omega + e.\emptyset \mid [\overline{a.c}.co \ k + \overline{e} \triangleright_k r] \]

\[ \xrightarrow{\text{R-Emb}} \quad [a.c.\omega + e.\emptyset \mid \overline{a.c}.co \ k + \overline{e} \triangleright_k a.c.\omega + e.\emptyset \mid r] \]

\[ \xrightarrow{\text{R-Comm}} \quad [\emptyset \triangleright_k a.c.\omega + e.\emptyset \mid r] \quad \text{(Deadlocked)} \]

\[ \xrightarrow{\text{R-Ab}} \quad a.c.\omega + e.\emptyset \mid r \quad \text{(The environment is restored)} \]

Simple Example (all traces)

\[ a.c.\omega + e.\emptyset \mid [\overline{a.c}.co \ k + \overline{e} \triangleright_k r] \]

\[ \xrightarrow{\text{R-Ab}} \quad a.c.\omega + e.\emptyset \mid r \]

Will never be sad: \( \emptyset \)

assuming \( r \) does not contain \( \overline{e} \)
**Aborting transactions**

A commit step makes the effects of the transaction permanent (**Durability**)

An abort step:
- restarts the transaction
- rolls-back embedded processes to their state before embedding (**Consistency**)
- does not roll-back actions that happened before embedding
- does not affect non-embedded processes

The behavioural theory will show the **Atomicity** property.

---

**Restarting transactions**

\[ a.c.\omega + e.\omega | \mu X. [\overline{a.e}.co k + e \triangleright_k X] \]

Will never be sad: \( \emptyset \)

---

*Infinitely aborting loop*
### Double Embedding

\[
\begin{align*}
[a.co \ k \ | \ b \ \triangleright_k 0] & \parallel [\bar{a}.co \ l \ | \ c \ \triangleright_l 0] \\
R-\text{EMB} & \rightarrow [a.co \ k \ | \ b \ \triangleright_k [\bar{a}.co \ l \ | \ c \ \triangleright_l 0]] \\
R-\text{EMB} & \rightarrow [b \ | \ [a.co \ k \ | \ \bar{a}.co \ l \ | \ c \ \triangleright_l 0]] \triangleright_k [\bar{a}.co \ l \ | \ c \ \triangleright_l 0] \\
R-\text{COMM} & \rightarrow [b \ | \ [c.o \ k \ | \ c \ \triangleright_k 0]] \triangleright_k [\bar{a}.co \ l \ | \ c \ \triangleright_l 0] \\
R-\text{CO} & \rightarrow [b \ | \ c \ \triangleright_k [\bar{a}.co \ l \ | \ c \ \triangleright_l 0]] \\
R-\text{CO} & \rightarrow [b \ | \ c]
\end{align*}
\]

### Safety properties

**Safety**: “Nothing bad will happen” [Lamport’77]

- A safety property can be formulated as a safety test \( T^\circ \) which signals on channel \( \circ \) when it detects the bad behaviour.

**Examples**:

- \( \mu X.(a.X + e.\circ) \) can not perform \( e \) while performing any sequence of \( a.s \)
- \( T^\circ = e.\circ | \bar{a}.\bar{b} \) can not perform \( e \) when \( a \) followed by \( b \) is offered.

- \( P \) passes the safety test \( T^\circ \) when \( P | T^\circ \) can not output on \( \circ \)
  - This is the negation of passing a “may test” [DeNicola-Hennessy’84]

**Examples**:

- \( l_3 = \mu X. [a.b.co \ k \ + \ \bar{e} \ \triangleright_k X] \) passes safety test \( T^\circ \)
- \( l_4 = \mu X. [a.b.co \ k \ | \ \bar{e} \ \triangleright_k X] \) does not pass safety test \( T^\circ \)
Safety

Definition (Basic Observable)
\[ P \downarrow_\circ \text{ iff there exists } P' \text{ such that } P \rightarrow^* P' \mid \circ \]

- Basic observable actions are *permanent*
- True: \[[a.b.co k \mid \bar{e} \rhd_k \theta] \mid (e.\circ \mid \bar{a} \bar{b}) \downarrow_\circ\]
- False: \[[a.b.co k + \bar{e} \rhd_k \theta] \mid (e.\circ \mid \bar{a} \bar{b}) \downarrow_\circ\]

Definition (\( P \) Passes safety test \( T^\circ \))
\( P \) cannot \( T^\circ \) when \( P \mid T^\circ \downarrow_\circ \)

Definition (Safety Preservation)
\( S \xrightarrow[\text{safe}]{\text{\_}} l \) when \( \forall T^\circ. \ S \text{ cannot } T^\circ \) implies \( l \text{ cannot } T^\circ \)

Safety preservation: Examples

\[ S_{ab} = \mu X. [a.b.co k \rhd_k X] \]
\[ I_3 = \mu X. [a.b.co k + \bar{e} \rhd_k X] \]
\[ I_4 = \mu X. [a.b.co k \mid \bar{e} \rhd_k X] \]

- \( S_{ab} \xrightarrow[\text{safe}]{\text{\_}} I_4 \) use test \( T^\circ = e.\circ \mid \bar{a} \bar{b} \)
- \( S_{ab} \xrightarrow[\text{safe}]{\text{\_}} I_3 \) - proof techniques required

- \( \tau.P + \tau.Q \xrightarrow[\text{safe}]{\text{\_}} [P \rhd_k Q] \), for any \( P, Q \) - proof techniques reqd
Liveness

**Liveness**: “Something good will eventually happen” [Lamport’77]

- A liveness property can be formulated as a *liveness test* $T^\omega$ which detects and reports good behaviour on $\omega$.

**Examples:**

- $T^\omega = \overline{a}b.\omega$ can do an $a$ then a $b$

- $\mu X. \left[ \overline{a}b.(\omega \mid co I) \triangleright_{I} X \right]$ can eventually do an $a$, $b$ uninterrupted?

- $a.\mu X. \left[ \overline{b}c.(\omega \mid co I) \triangleright_{I} X \right]$ English?

- $P$ **passes the liveness test** $T^\omega$ when $\omega$ is eventually guaranteed

**Dilemma**: What does this mean?

---

**Dilemma**

Does $\mu X. \left[ a.b.co k \triangleright_{k} X \right]$ pass liveness test $T^\omega_{ab} = \overline{a}b.\omega$?

- **must-testing**: NO because of infinite loop
- **should-testing**: YES
Liveness testing

Definition (P Passes liveness test $T^ω$ [Rensink-Vogler'07])

\[ P \text{ shd } T^ω \text{ when } \forall R. \ P \mid T^ω \rightarrow^* R \implies R \downarrow^ω \]

Examples:

- $\mu X. \llbracket a.b.co k \triangleright_k X \rrbracket$ passes liveness test $T^ω_{ab} = a.b.ω$

- $\llbracket a.b.co k \triangleright_k \emptyset \rrbracket$ does not pass test $T^ω_{ab}$

Definition (Liveness preservation)

\[ S \overset{\sim}_{\text{live}} \llbracket I \rrbracket \text{ when } \forall T^ω. \ S \text{ shd } T^ω \implies I \text{ shd } T^ω \]

Liveness preservation: Examples

\[ S_{ab} = \mu X. \llbracket a.b.co k \triangleright_k X \rrbracket \]
\[ I_2 = \mu X. \llbracket a.b.\emptyset \triangleright_k X \rrbracket \]
\[ I_3 = \mu X. \llbracket a.b.co k + ε \triangleright_k X \rrbracket \]

- $S_{ab} \overset{\sim}{\sim}_{\text{live}} I_2$ use test $T^ω = a.b.ω$

- $S_{ab} \overset{\sim}{\sim}_{\text{live}} I_3$ – proof techniques required

- $\mu X. \llbracket P \mid \text{co } k \triangleright_k X \rrbracket \overset{\sim}{\sim}_{\text{live}} P$, for any $P$ – proof techniques rqd

Proof techniques:

Require characterisations using “traces” and “refusals”
Compositional Semantics

- The embedding rule is simple but entangles the processes
- We need to reason about the behaviour of $P|Q$ in terms of $P$ and $Q$
- We introduce a compositional Labelled Transition System that uses *secondary transactions*: $\llbracket P \triangleright_k Q \rrbracket^o$

Compositional Semantics: safe-testing

The behaviour of processes in TransCCS can be understood by a *simple subset of the LTS traces*:

- where *all actions are eventually committed*
- that *ignore transactional annotations* on the traces

$$\text{Tr}_{\text{clean}}\left(\llbracket a.c.co \ k \triangleright_k e \rrbracket\right) = \{\epsilon, \ a, \ c, \ e\}$$

$$\text{Tr}_{\text{clean}}\left(\mu X. \llbracket a.c.co \ k \triangleright_k X \rrbracket\right) = \{\epsilon, \ a, \ c\}$$

- Set of clean traces not prefix closed: *atomicity*

Characterisation of Safe Testing:

$$P \sqsubseteq_{\text{safe}} Q \iff \text{Tr}_{\text{clean}}(P) \subseteq \text{Tr}_{\text{clean}}(Q)$$

- To understand the safe-testing behaviour of $P$ we only need to consider the clean traces $\text{Tr}_{\text{clean}}(P)$. 
Compositional semantics: should-testing

Tree Failures: [Rensink-Vogler’07]

\((t, \text{Ref})\) where
- \(t\) is a clean trace
- \(\text{Ref}\) is a set of clean traces

Tree failures of a process:
\((t, \text{Ref})\) is a tree failure of \(P\) when
\[\exists P'. \quad P \Rightarrow_{\text{CL}} P' \quad \text{and} \quad \mathcal{L}(P') \cap \text{Ref} = \emptyset\]

\[\mathcal{F}(P) = \{(t, \text{Ref}) \text{ tree failure of } P\}\]

Characterisation of should-testing:
\[S \not\subseteq \text{live } I \iff \mathcal{F}(S) \supseteq \mathcal{F}(I)\]

Simple Examples

Let \(S_{ab} = \mu X. [a.b.co k \triangleright_k X]\)

\[\mathcal{L}(S_{ab}) = \{\epsilon, ab\}\]

\[\mathcal{F}(S_{ab}) = \{(\epsilon, S \backslash ab), (ab, S) \mid S \subseteq A^*\}\]

- \(S_{ab} \simeq_{\text{safe}} I_1 = [a.b.co k \triangleright_k \emptyset]\)

\[\mathcal{L}(I_1) = \{\epsilon, ab\}\]

\[\mathcal{F}(I_1) = \{(\epsilon, S), (ab, S) \mid S \subseteq A^*\}\]

- \(S_{ab} \simeq_{\text{live}} I_2 = \mu X. [a.b.co k + e \triangleright_k X]\)

\[\mathcal{L}(I_2) = \mathcal{L}(S_{ab})\]

\[\mathcal{F}(I_2) = \mathcal{F}(S_{ab})\]
Summary

- **TransCCS**: a language for communicating/co-operative transactions
- simple reduction semantics using an *embedding* rule
- behavioural theories for preservation of
  - safety properties
  - liveness properties
- characterisations which allow
  - proofs of equivalences
  - equational laws

References:
- *Communicating Transactions*, Concur 2010
- *Liveness of Communicating Transactions*, APLAS 2010

Current work:
- Extension to Haskell/CML
- prototype implementation
- Proof techniques based on traces, refusal trees, co-induction

THANK YOU!
Workshop announcement

1st Workshop on Optimistic Cooperation in Concurrent Programming (OCCP 2013)

- Location: Rome, Italy (co-located with ETAPS 2013)
- Date: Saturday March 16th, 2013
- Submissions: 14th Dec (abstracts) 21st Dec (Papers)