A theory of communicating transactions

Matthew Hennessy

joint work with Edsko de Vries, Vasileois Koutavas

CHoCoLa, PPS Paris December 2012

TRINITY COLLEGE DUBLIN
COLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH

Outline

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

$$P_1 \,||\, P_2 \,||\, P_3 \,||\, P_4$$

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

$$P_1 \,||\, P_2 \,||\, P_3 \,||\, P_4$$

Algorithm for P_n:

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

Example: programming a three-way rendezvous

$$P_1 \,||\, P_2 \,||\, P_3 \,||\, P_4$$

$$
\begin{aligned}
P_n \quad \Leftarrow \quad & b!\langle n\rangle \mid b!\langle n\rangle \mid \\
& \text{atomic}[\![b?(y)\,.b?(z)\,. \\
& \quad s_y!\langle n,z\rangle\,.s_z!\langle n,y\rangle\,. \qquad \text{proposing} \\
& \quad s_n?(y_1,z_1)\,.s_n?(y_2,z_2)\,. \qquad \text{confirming} \\
& \quad \texttt{if}\ \{y,z\} = \{y_1,z_1\} = \{y_2,z_2\} \\
& \quad \texttt{then}\ \mathit{commit} \mid \texttt{out}_n!\langle y,z\rangle \\
& \quad \texttt{else}\ \text{abrt\&retry}\ ]\!]
\end{aligned}
$$

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

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

Transaction $[\![P \triangleright_k Q]\!]$

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

Rollbacks and Commits

Co-operating actions: $a \leftarrow$ needs co-operation of $\rightarrow \overline{a}$

$$T_a \mid T_b \mid T_c \mid P_d \mid P_e$$

where

$$
\begin{array}{lcl}
T_a & = & [\![\overline{d}.\overline{b}.(\mathtt{co}\ k_1 \mid a) \triangleright_{k_1} \mathbf{0}]\!] \\
T_b & = & [\![\overline{c}.(\mathtt{co}\ k_2 \mid b) \triangleright_{k_2} \mathbf{0}]\!] \\
T_c & = & [\![\overline{e}.c.\mathtt{co}\ k_3 \triangleright_{k_3} \mathbf{0}]\!] \\
P_d & = & d.R_d \\
P_e & = & e.R_e
\end{array}
$$

- 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
$$\frac{a_i = \overline{b_j}}{\sum_{i \in I} a_i.P_i \mid \sum_{j \in J} b_j.Q_j \to P_i \mid Q_j}$$
Communication

R-Co
$$\frac{}{[\![P \mid \text{co } k \rhd_k Q]\!] \to P}$$
Commit

R-Ab
$$\frac{}{[\![P \rhd_k Q]\!] \to Q}$$
Random abort

R-Emb
$$\frac{k \notin R}{[\![P \rhd_k Q]\!] \mid R \to [\![P \mid R \rhd_k Q \mid R]\!]}$$
Embed

Simple Example

Convention:

- ω: I am happy
- $\mho$: I am sad

$$a.c.\omega + e.\mho \mid [\![\overline{a}.\overline{c}.\text{co } k + \overline{e} \rhd_k r]\!]$$
$$\xrightarrow{\text{R-Emb}} [\![a.c.\omega + e.\mho \mid \overline{a}.\overline{c}.\text{co } k + \overline{e} \rhd_k a.c.\omega + e.\mho \mid r]\!]$$
$$\xrightarrow{\text{R-Comm}} [\![c.\omega \mid \overline{c}.\text{co } k \rhd_k a.c.\omega + e.\mho \mid r]\!]$$
$$\xrightarrow{\text{R-Comm}} [\![\omega \mid \text{co } k \rhd_k a.c.\omega + e.\mho \mid r]\!]$$
$$\xrightarrow{\text{R-Co}} \omega$$

Simple Example (a second trace)

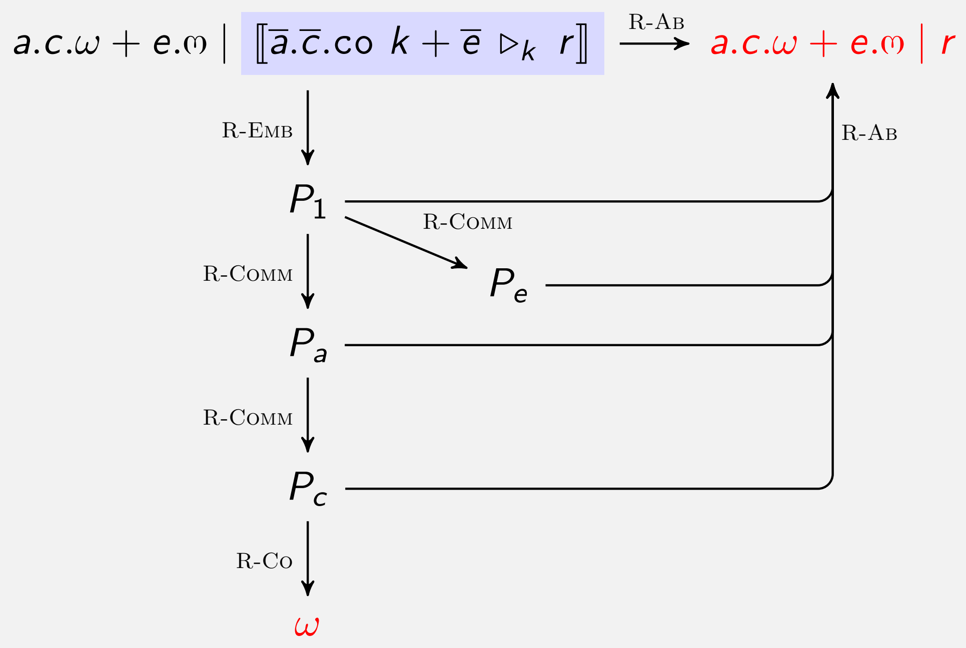
$$a.c.\omega + e.\frownie \mid [\![\overline{a}.\overline{c}.\mathrm{co}\ k + \overline{e} \rhd_k\ r]\!]$$

$$\xrightarrow{\text{R-EMB}} [\![a.c.\omega + e.\frownie \mid \overline{a}.\overline{c}.\mathrm{co}\ k + \overline{e} \rhd_k\ a.c.\omega + e.\frownie \mid r]\!]$$

$$\xrightarrow{\text{R-COMM}} [\![\quad \frownie \quad \rhd_k\ a.c.\omega + e.\frownie \mid r]\!] \quad \text{(Deadlocked)}$$

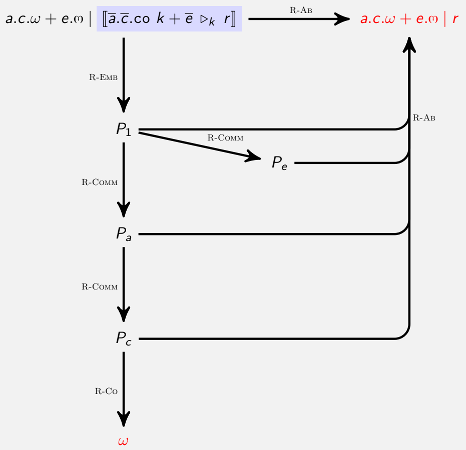
$$\xrightarrow{\text{R-AB}} a.c.\omega + e.\frownie \mid r \quad \text{(The environment is restored)}$$

Simple Example (all traces)



Will never be sad: $\frownie$ assuming r does not contain $\overline{e}$

Aborting transactions



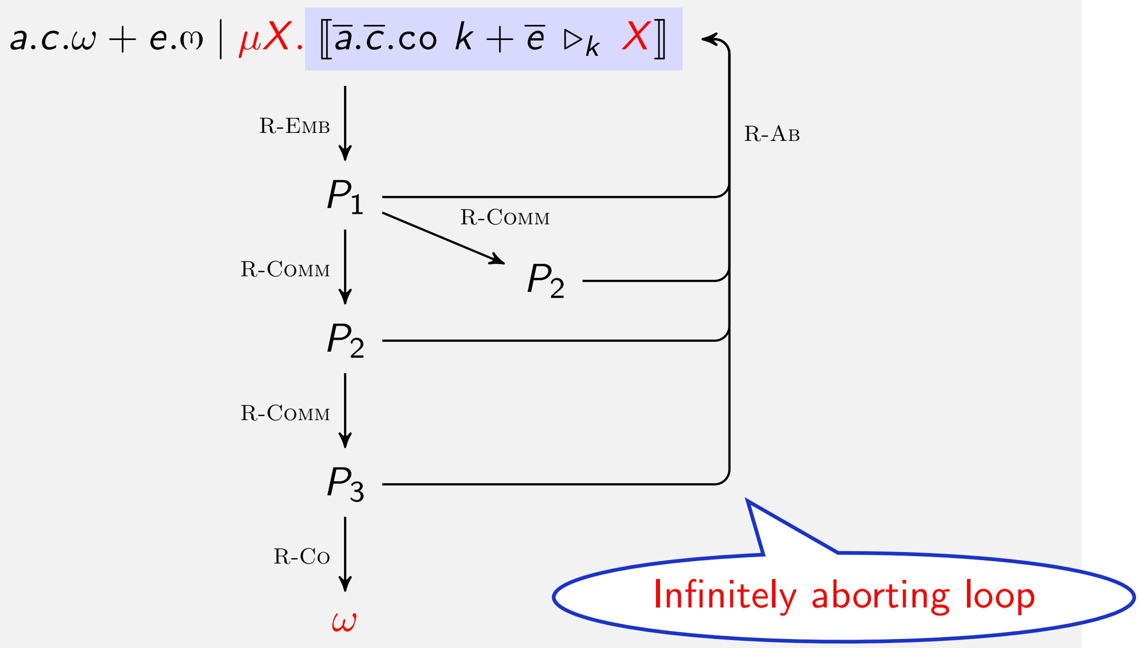
A commit step makes the effects of the transaction permanent (**Durability**)

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



Will never be sad: ☹

Double Embedding

$$[\![a.\mathrm{co}\ k \mid b \rhd_k \mathbf{0}]\!] \mid [\![\overline{a}.\mathrm{co}\ l \mid c \rhd_l \mathbf{0}]\!]$$

$$\xrightarrow{\text{R-Emb}} \Big[\!\!\Big[a.\mathrm{co}\ k \mid b \mid [\![\overline{a}.\mathrm{co}\ l \mid c \rhd_l \mathbf{0}]\!] \rhd_k [\![\overline{a}.\mathrm{co}\ l \mid c \rhd_l \mathbf{0}]\!] \Big]\!\!\Big]$$

$$\xrightarrow{\text{R-Emb}} \Big[\!\!\Big[b \mid [\![a.\mathrm{co}\ k \mid \overline{a}.\mathrm{co}\ l \mid c \rhd_l a.\mathrm{co}\ k]\!] \rhd_k [\![\overline{a}.\mathrm{co}\ l \mid c \rhd_l \mathbf{0}]\!] \Big]\!\!\Big]$$

$$\xrightarrow{\text{R-Comm}} \Big[\!\!\Big[b \mid [\![\mathrm{co}\ k \mid \mathrm{co}\ l \mid c \rhd_l a.\mathrm{co}\ k]\!] \rhd_k [\![\overline{a}.\mathrm{co}\ l \mid c \rhd_l \mathbf{0}]\!] \Big]\!\!\Big]$$

$$\xrightarrow{\text{R-Co}} \Big[\!\!\Big[b \mid \mathrm{co}\ k \mid c \rhd_k [\![\overline{a}.\mathrm{co}\ l \mid c \rhd_l \mathbf{0}]\!] \Big]\!\!\Big]$$

$$\xrightarrow{\text{R-Co}} b \mid c$$

Safety properties

Safety: "Nothing bad will happen" [Lamport'77]

- A safety property can be formulated as a *safety test* T^{ω} which signals on channel ω when it detects the bad behaviour

Examples:

  - $\mu X.(a.X + e.\omega)$ can not perform e while performing any sequence of as
  - $T^{\omega} = e.\omega \mid \overline{a}.\overline{b}$ can not perform e when a followed by b is offered.

- *P passes the safety test T^{ω}* when $P \mid T^{\omega}$ can not output on ω
  - This is the negation of passing a "may test" [DeNicola-Hennessy'84]

Examples:

- $I_3 = \mu X.\, [\![a.b.\mathrm{co}\ k + \overline{e} \rhd_k X]\!]$ passes safety test T^{ω}
- $I_4 = \mu X.\, [\![a.b.\mathrm{co}\ k \mid \overline{e} \rhd_k X]\!]$ does not pass safety test T^{ω}

Safety

Definition (Basic Observable)

$P \Downarrow_\omega$ iff there exists P' such that $P \to^* P' \mid \omega$

- Basic observable actions are *permanent*
- True: $[\![a.b.\mathsf{co}\ k \mid \overline{e} \rhd_k \mathbf{0}]\!] \;\mid\; (e.\omega \mid \overline{a}.\overline{b}) \Downarrow_\omega$
- False: $[\![a.b.\mathsf{co}\ k + \overline{e} \rhd_k \mathbf{0}]\!] \;\mid\; (e.\omega \mid \overline{a}.\overline{b}) \Downarrow_\omega$

Definition (P Passes safety test T^ω)

$$P \;\mathrm{cannot}\; T^\omega \quad \text{when} \quad P \mid T^\omega \not\Downarrow_\omega$$

Definition (Safety Preservation)

$$S \sqsubseteq_{\mathrm{safe}} I \quad \text{when} \quad \forall T^\omega.\quad S \;\mathrm{cannot}\; T^\omega \quad \text{implies} \quad I \;\mathrm{cannot}\; T^\omega$$

Safety preservation: Examples

$$S_{ab} = \mu X.\, [\![a.b.\mathsf{co}\ k \rhd_k X]\!]$$
$$I_3 = \mu X.\, [\![a.b.\mathsf{co}\ k + \overline{e} \rhd_k X]\!]$$
$$I_4 = \mu X.\, [\![a.b.\mathsf{co}\ k \mid \overline{e} \rhd_k X]\!]$$

- $S_{ab} \not\sqsubseteq_{\mathrm{safe}} I_4$ use test $T^\omega = e.\omega \mid \overline{a}.\overline{b}$
- $S_{ab} \sqsubseteq_{\mathrm{safe}} I_3$ – proof techniques required
- $\tau.P + \tau.Q \sqsubseteq_{\mathrm{safe}} [\![P \rhd_k Q]\!]$, for any P, Q – proof techniques rqd

Liveness

Liveness: "Something good will eventually happen" [Lamport'77]

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

Examples:

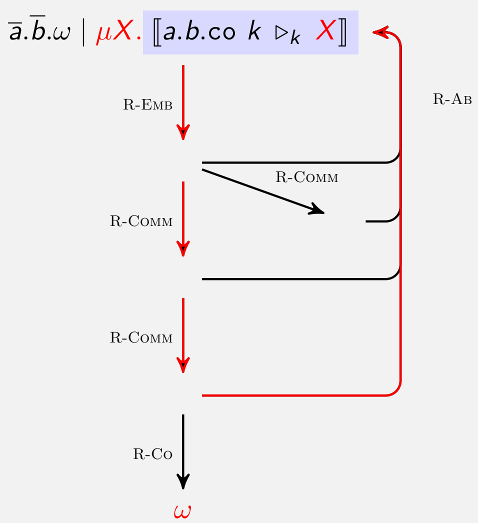
- $T^\omega = \overline{a}.\overline{b}.\omega$ can do an a then a b
- $\mu X. \llbracket \overline{a}.\overline{b}.(\omega \mid \text{co } l) \triangleright_l X \rrbracket$ can eventually do an a, b uninterrupted?
- $a.\mu X. \llbracket \overline{b}.\overline{c}.(\omega \mid \text{co } l) \triangleright_l X \rrbracket$ English?

- *P passes the liveness test* T^ω when ω is eventually guaranteed

Dilemma: What does this mean?

Dilemma

Does $\mu X. \llbracket a.b.\text{co } k \triangleright_k X \rrbracket$ pass liveness test $T^\omega_{ab} = \overline{a}.\overline{b}.\omega$?



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

Liveness testing

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

$$P \,\mathrm{shd}\, T^\omega \quad \text{when} \quad \forall R. \quad P \mid T^\omega \to^* R \quad \text{implies} \quad R{\Downarrow_\omega}$$

Examples:

- $\mu X.\, [\![a.b.\mathsf{co}\ k \rhd_k X]\!]$ passes liveness test $T^\omega_{ab} = \overline{a}.\overline{b}.\omega$
- $[\![a.b.\mathsf{co}\ k \rhd_k \mathbb{0}]\!]$ does not pass test T^ω_{ab}

Definition (Liveness preservation)

$$S \sqsubseteq_{\mathrm{live}} I \quad \text{when} \quad \forall T^\omega. \quad S \,\mathrm{shd}\, T^\omega \quad \text{implies} \quad I \,\mathrm{shd}\, T^\omega$$

Liveness preservation:Examples

$$\begin{aligned} S_{ab} &= \mu X.\, [\![a.b.\mathsf{co}\ k \rhd_k X]\!] \\ I_2 &= \mu X.\, [\![a.b.\mathbb{0} \rhd_k X]\!] \\ I_3 &= \mu X.\, [\![a.b.\mathsf{co}\ k + \overline{e} \rhd_k X]\!] \end{aligned}$$

- $S_{ab} \not\sqsubseteq_{\mathrm{live}} I_2$ use test $T^\omega = \overline{a}.\overline{b}.\omega$
- $S_{ab} \sqsubseteq_{\mathrm{live}} I_3$ – proof techniques required
- $\mu X.\, [\![P \mid \mathsf{co}\ k \rhd_k X]\!] \eqsim_{\mathrm{live}} P$, 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*: $[\![P \rhd_k Q]\!]^\circ$

$$a.c.\mathbf{0} \xrightarrow{\text{emb } k} [\![a.c.\mathbf{0} \rhd_k a.c.\mathbf{0}]\!]^\circ \xrightarrow{k(a)} [\![c.\mathbf{0} \rhd_k a.c.\mathbf{0}]\!]^\circ \xrightarrow{k(c)} [\![\mathbf{0} \rhd_k a.c.\mathbf{0}]\!]^\circ \xrightarrow{\text{ab } k} a.c.\mathbf{0} \xrightarrow{\text{co } k} \mathbf{0}$$

$$[\![\bar{a}.\bar{c}.\text{co } k \mid \bar{a}.\mathbf{0} \rhd_k e]\!] \xrightarrow{\text{emb } k} [\![\bar{a}.\bar{c}.\text{co } k \mid \bar{a}.\mathbf{0} \rhd_k e]\!] \xrightarrow{k(\bar{a})} [\![\bar{c}.\text{co } k \mid \bar{a}.\mathbf{0} \rhd_k e]\!] \xrightarrow{k(\bar{c})} [\![\text{co } k \mid \bar{a}.\mathbf{0} \rhd_k e]\!] \xrightarrow{\text{ab } k} e \xrightarrow{\text{co } k} \bar{a}.\mathbf{0}$$

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

$$\text{Tr}_{\text{clean}}\left([\![a.c.\text{co } k \rhd_k e]\!] \right) = \{\epsilon,\ \mathbf{a\,c},\ \mathbf{e}\}$$

$$\text{Tr}_{\text{clean}}\left(\mu X.\, [\![a.c.\text{co } k \rhd_k X]\!] \right) = \{\epsilon,\ \mathbf{a\,c}\}$$

- Set of clean traces not prefix closed: atomicity

Characterisation of Safe Testing:

$$P \sqsubset\!\!\!\!\sim_{\text{safe}} Q \qquad \text{iff} \qquad \text{Tr}_{\text{clean}}(P) \subseteq \text{Tr}_{\text{clean}}(Q)$$

- 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, Ref) where

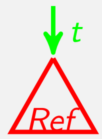
- t is a clean trace
- Ref is a set of clean traces

can be non-prefixed closed

Tree failures of a process:

(t, Ref) is a tree failure of P when

$$\exists P'. \quad P \stackrel{t}{\Rightarrow}_{CL} P' \quad \text{and} \quad \mathcal{L}(P') \cap Ref = \emptyset$$



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

Characterisation of should-testing:

$$S \sqsubseteq_{\text{live}} I \quad \text{iff} \quad \mathcal{F}(S) \supseteq \mathcal{F}(I)$$

Simple Examples

Let $\quad S_{ab} = \mu X. \llbracket a.b.\text{co } k \rhd_k X \rrbracket$ $\qquad \mathcal{L}(S_{ab}) = \{\epsilon, ab\}$

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

- $S_{ab} \eqsim_{\text{safe}} I_1 = \llbracket a.b.\text{co } k \rhd_k \mathbb{0} \rrbracket$ $\qquad \mathcal{L}(I_1) = \{\epsilon, ab\}$
  $S_{ab} \not\sqsubseteq_{\text{live}} I_1$ $\qquad \mathcal{F}(I_1) = \{(\epsilon, S), (ab, S) \mid S \subseteq A^*\}$
- $S_{ab} \eqsim_{\text{safe}} I_2 = \mu X. \llbracket a.b.\text{co } k + e \rhd_k X \rrbracket$ $\qquad \mathcal{L}(I_2) = \mathcal{L}(S_{ab})$
  $S_{ab} \eqsim_{\text{live}} I_2$ $\qquad \mathcal{F}(I_2) = \mathcal{F}(S_{ab})$

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)

Details: http://www.cs.tcd.ie/Vasileios.Koutavas/occp-workshop