Communicating Transactions

a survey

Matthew Hennessy

joint work with Edsko de Vries, Vasileois Koutavas

WG 2.2, Amsterdam, September 2012

TRINITY COLLEGE DUBLIN
COLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH

Outline

A workshop

Transactions

Co-operating Transactions

TransCCS

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

---

Database 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

Multiple transactions run

- concurrently
- optimistically: hoping no interference will occur

STM: Software Transactional Memory

- Database technology applied to software
- concurrency control: *atomic memory transactions*
- lock-free programming in multithreaded programmes
- threads run optimistically
- conflicts are automatically rolled back by system

Implementations:

- Haskell, OCaml
- C,C++,Csharp
- Java, Scala
- Intel Haswell architecture
- . . .

STM: An example

$$\text{atomic}\,[\![P]\!]\ ||\ \text{atomic}\,[\![Q]\!]$$

- P: $y ::= x;\ y ::= y + 1;\ x ::= y;\ y ::= 0$
- Q: $y ::= x;\ y ::= y + 2;\ x ::= y;\ y ::= 0$

Result: x increased by

- 3
- not 0

Issues:

- Language Design
- Implementation strategies
- Semantics what should happen when programs are run

Standard Transactions

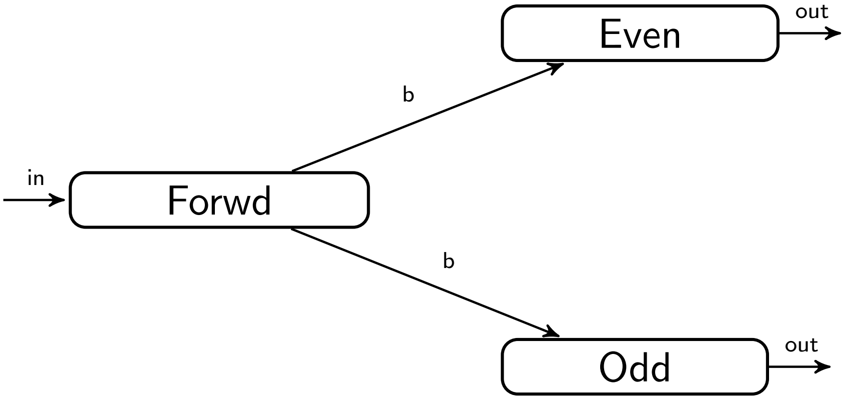
- Transactions provide *an abstraction for error recovery* in a concurrent setting.
- Guarantees:
  - Atomicity: Each transaction either runs in its entirety (commits) or not at all
  - Consistency: When faults are detected the transaction is automatically rolled-back
  - Isolation: The effects of a 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)

An example



$$\mathrm{Forwd} \Leftarrow \mathrm{in}?(x)\,.\,b!\langle x\rangle\,.\mathrm{Forwd}$$

$$\begin{aligned}\mathrm{Even} \Leftarrow \mathrm{atomic}\llbracket & b?(x)\,.\,\mathtt{if}\ \mathrm{even}(x)\ \mathtt{then}\ \mathrm{out}!\langle f(x)\rangle\,.(\mathit{commit} \mid \mathrm{Even}) \\ & \mathtt{else}\ \mathrm{abrt\&retry}\rrbracket\end{aligned}$$

$$\begin{aligned}\mathrm{Odd} \Leftarrow \mathrm{atomic}\llbracket & b?(x)\,.\,\mathtt{if}\ \mathrm{Odd}(x)\ \mathtt{then}\ \mathrm{out}!\langle g(x)\rangle\,.(\mathit{commit} \mid \mathrm{Odd}) \\ & \mathtt{else}\ \mathrm{abrt\&retry}\rrbracket\end{aligned}$$

Example: three-way rendezvous

$$P_1 \mid\mid P_2 \mid\mid P_3 \mid\mid P_4$$

Problem:

- P_i process/transaction subject to failure
- Some three P_i should decide to collaborate

Result:

- Each P_j in the coalition outputs id of its partners on channel out_j

Example: three-way rendezvous

$$P_1 \mid\mid P_2 \mid\mid P_3 \mid\mid P_4$$

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: three-way rendezvous

$$P_1 \mid\mid P_2 \mid\mid P_3 \mid\mid P_4$$

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

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 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:

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

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

- execute P to completion (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: $\boxed{a \longleftarrow \text{needs co-operation of} \longrightarrow \overline{a}}$

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

where

$$\begin{array}{rcl}
T_a & = & \llbracket \overline{d}.\overline{b}.(\mathsf{co}\ k_1 \mid a) \triangleright_{k_1} \mathbf{0} \rrbracket \\
T_b & = & \llbracket \overline{c}.(\mathsf{co}\ k_2 \mid b) \triangleright_{k_2} \mathbf{0} \rrbracket \\
T_c & = & \llbracket \overline{e}.c.\mathsf{co}\ k_3 \triangleright_{k_3} \mathbf{0} \rrbracket \\
P_d & = & d.R_d \\
P_e & = & e.R_e
\end{array}$$

- if T_c aborts, what roll-backs are necessary?
- When can action a be considered permanent?
- When can code P_d be considered permanent?

Reduction semantics main rules

R-Comm

$$\dfrac{a_i = \overline{b_j}}{\sum_{i \in I} a_i.P_i \mid \sum_{j \in J} b_j.Q_j \rightarrow P_i \mid Q_j}$$

Communication

R-Co

$$\dfrac{}{\llbracket P \mid \mathsf{co}\ k \triangleright_k Q \rrbracket \rightarrow P}$$

Commit

R-Ab

$$\dfrac{}{\llbracket P \triangleright_k Q \rrbracket \rightarrow Q}$$

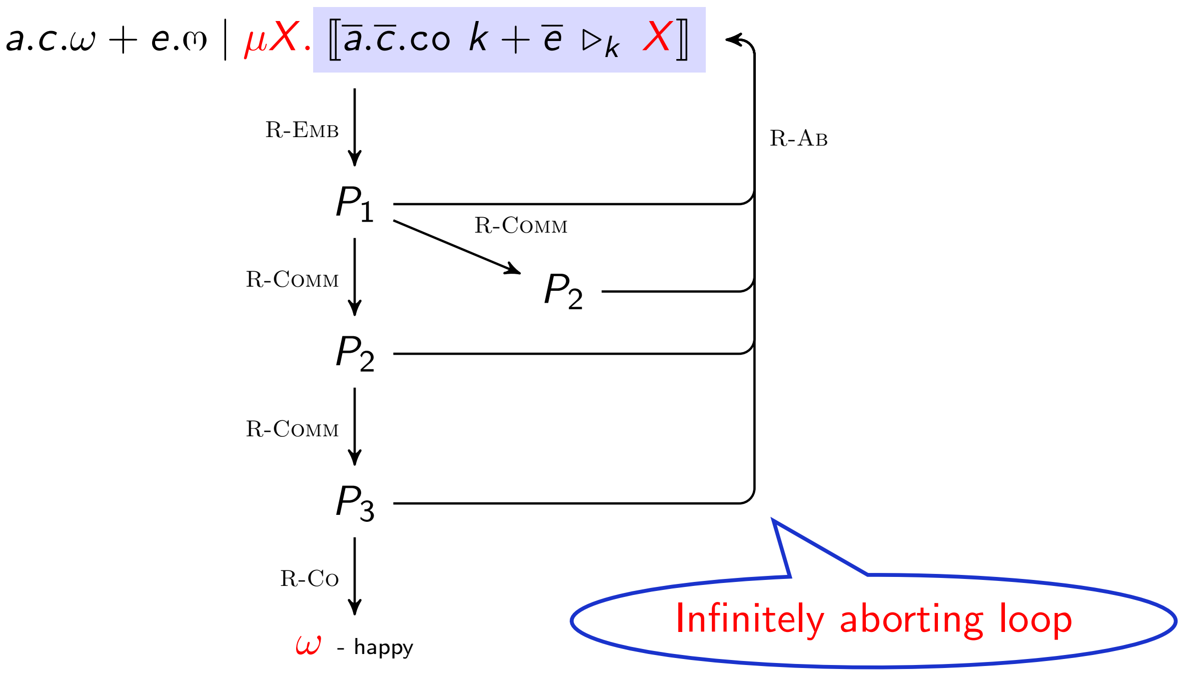
Random abort

R-Emb

$$\dfrac{k \notin R}{\llbracket P \triangleright_k Q \rrbracket \mid R \rightarrow \llbracket P \mid R \triangleright_k Q \mid R \rrbracket}$$

Embed

Example: restarting transactions



Will never be sad: $\mho$

Safety properties

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

- A safety property can be formulated as a *safety test* $T^{\mho}$ which signals on channel $\mho$ when it detects the bad behaviour
- *P passes the safety test $T^{\mho}$* when $P \mid T^{\mho}$ can not output on $\mho$
  - This is the negation of passing a "may test" [DeNicola-Hennessy'84]

Definition (Safety Preservation)

$$S \mathrel{\underset{\sim}{\sqsubset}}_{\mathrm{safe}} I \quad \text{when} \quad \forall T^{\mho}.\quad S \,\mathrm{cannot}\, T^{\mho} \quad \text{implies} \quad I \,\mathrm{cannot}\, T^{\mho}$$

Safety preservation: Examples

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

- $S_{ab} \not\mathrel{\underset{\sim}{\sqsubset}}_{\text{safe}} I_4$ use test $T^{\mho} = e.\mho \mid \overline{a}.\overline{b}$
- $S_{ab} \mathrel{\underset{\sim}{\sqsubset}}_{\text{safe}} I_3$ – proof techniques required
- $\tau.P + \tau.Q \mathrel{\underset{\sim}{\sqsubset}}_{\text{safe}} [\![P \rhd_k Q]\!]$, 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.
- *P passes the liveness test* T^{ω} when ω is eventually guaranteed

  What does this mean? $P \operatorname{shd} T^{\mho}$

Definition (Liveness preservation)

$$S \mathrel{\underset{\sim}{\sqsubset}}_{\text{live}} I \quad \text{when} \quad \forall T^{\omega}. \quad S \operatorname{shd} T^{\mho} \quad \text{implies} \quad I \operatorname{shd} T^{\omega}$$

Liveness preservation:Examples

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

- $S_{ab} \underset{\sim}{\not\sqsubset}_{\text{live}} I_2$ use test $T^{\omega} = \overline{a}.\overline{b}.\omega$
- $S_{ab} \underset{\sim}{\sqsubset}_{\text{live}} I_3$ – proof techniques required
- $\mu X.\, [\![P \mid \mathrm{co}\ k \rhd_k X]\!] \eqsim_{\text{live}} P$, for any P

  – proof techniques rqd

Proof techniques:

Require characterisations using "traces" and "refusals"

Results

Characterisation of Safe Testing:

$$P \underset{\sim}{\sqsubset}_{\mathrm{may}} Q \qquad \text{iff} \qquad \mathrm{Tr}_{\mathrm{clean}}(P) \subseteq \mathrm{Tr}_{\mathrm{clean}}(Q)$$

$\mathrm{Tr}_{\mathrm{clean}}(R)$:

- sequences of communications performed by R which are *eventually committed*
- $\mathrm{Tr}_{\mathrm{clean}}\left(\mu X.\, [\![a.c.\mathrm{co}\ k + e \rhd_k X]\!] \right) = \{\epsilon,\ \mathbf{a\,c}\}$
- non-prefixed closed in general

Characterisation of should-testing:

$$S \underset{\sim}{\sqsubset}_{\text{live}} I \quad \text{iff} \quad \mathcal{F}(S) \supseteq \mathcal{F}(I)$$

$\mathcal{F}(P)$: generalisation of CSP refusals/failures

Some references

- Edsko de Vries, Vasileios Koutavas and Matthew Hennessy. *Liveness of Communicating Transactions*, Proceedings of APLAS, 2010.
- Mohsen Lesani, Jens Palsberg. *Communicating memory transactions*, Proceedings of Principles and Practice of Parallel Programming, 2011.
- Kevin Donnelly, Matthew Fluet. *Transactional events*, Journal of Functional Programming, 2008.
- Tim Harris, Simon Marlow, Simon L. Peyton Jones, Maurice Herlihy. *Composable memory transactions*, Communications of ACM, 2008.