Co-operating/Communicating Transactions

a survey

Matthew Hennessy

joint work with Edsko de Vries, Vasileois Koutavas, Carlo Spaccasassi

Bertinoro, June 2014
Outline

Background

Co-operating Transactions

TransCCS
First software crisis

- Transistor: 1947
- Intel 4004: 1971
- Pentium 4: 2000

CPU performance

- Software Crisis
- Explosion in high-level programming languages
  - Precursors: Fortran, Algol, Lisp
  - Procedural (C, Pascal, Ada, ...)
  - Object-Oriented (C++, Java, Eiffel, ...)
  - Functional (ML, Scheme, Haskell, ...)
  - Declarative (Prolog, SQL, ...)

New building blocks for software
Second software crisis

Transistor 1947
Intel 4004 1971
Pentium 4 2000

CPU performance

2nd Software Crisis
Second software crisis reasons

[“The Future of Computing Performance: Game Over or Next Level?”
Multi-core Programming Technology

Precursors

- Multiple threads + shared memory access + locks single-core concurrency
- ...

New building blocks

- Algorithmic skeletons
  - Software design patterns for *easily parallelisable tasks* MapReduce from Google, TBB from Intel, GCD from Apple, ...
- Software transactional memory
  - PL abstraction for hard multi-core problems: shared memory access
- Co-operating transactions This talk
  - PL abstraction for hard multi-core problems: concurrent consensus
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, Csharp, Intel Haswell architecture

Issues:
- Language Design
- Implementation strategies
- Semantics what should happen when programs are run
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, Csharp, Intel Haswell architecture

Issues:

- Language Design
- Implementation strategies
- Semantics what should happen when programs are run
Standard Transactions on which STM is based

- 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
Standard Transactions on which STM is based

- 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 co-operation/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

\[
\text{Forwd} \leftarrow \text{in}(x) \cdot b!(x) \cdot \text{Forwd}
\]

\[
\text{Even} \leftarrow \text{atomic}[b?(x) \cdot \text{if even}(x) \text{ then out!}(f(x)) \cdot (\text{commit} | \text{Even}) \text{ else abrt&retry}]
\]

\[
\text{Odd} \leftarrow \text{atomic}[b?(x) \cdot \text{if odd}(x) \text{ then out!}(g(x)) \cdot (\text{commit} | \text{Odd}) \text{ else abrt&retry}]
\]
An example

\[
\text{Forwd} \leftarrow \text{in}(x) \cdot b!\langle x \rangle \cdot \text{Forwd}
\]

\[
\text{Even} \leftarrow \text{atomic}\left[ b?(x) \cdot \text{if even}(x) \text{ then out!}\langle f(x) \rangle \cdot (\text{commit} \mid \text{Even}) \right. \\
\left. \text{ else abrt&retry} \right]
\]

\[
\text{Odd} \leftarrow \text{atomic}\left[ b?(x) \cdot \text{if odd}(x) \text{ then out!}\langle g(x) \rangle \cdot (\text{commit} \mid \text{Odd}) \right. \\
\left. \text{ else abrt&retry} \right]
\]
Example: three-way rendezvous

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

Problem:
- \( P_i \) process/transaction subject to failure
- Some coalition of three from \( P_1, P_2, P_3, P_4 \) should decide to collaborate

Result:
- Each \( P_j \) in the successful coalition outputs id of its partners on channel out\( j \)
Example: three-way rendezvous

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

Problem:
- \( P_i \) process/transaction subject to failure
- Some coalition of three from \( P_1, P_2, P_3, P_4 \) should decide to collaborate

Result:
- Each \( P_j \) in the successful coalition outputs id of its partners on channel \( \text{out}_j \)
Example: 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 \( b!\langle n \rangle \parallel b!\langle n \rangle \)
- 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

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

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 \( b?(y) \cdot b?(z) \)

- Propose coalition with these partners \( s_y!\langle n, z \rangle \cdot s_z!\langle n, y \rangle \)

- Confirm that partners are in agreement:
  - if YES, commit and report
  - if NO, abort\&retry
Example: three-way rendezvous

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

\[ P_n \leftarrow b!\langle n \rangle \mid b!\langle n \rangle \mid \]
atomic\( [ b?(y) . b?(z) .
\]
\[ s_y !\langle n, z \rangle . s_z !\langle n, y \rangle . \quad \text{proposing} \]
\[ s_n ?(y_1, z_1) . s_n ?(y_2, z_2) . \quad \text{confirming} \]
\[ \text{if } \{ y, z \} = \{ y_1, z_1 \} = \{ y_2, z_2 \} \]
\[ \text{then } \text{commit} \mid \text{out}_n !\langle y, z \rangle \]
\[ \text{else } \text{abrt&retry} \]
Co-operating Transactions: Issues

- **Language Design**
  - Transaction Synchronisers (Luchangco et al 2005)
  - cJoin with commits Bruni, Melgratti, Montanari ENTCS 2004
  - Transactional Events for ML (Fluet, Grossman et al. ICFP 2008)
  - Communication Memory Transactions (Lesani, Palsberg PPoPP 2011)
  - ... Abstractions for Concurrent Consensus (Spaccasassi, Koutavas, Trends in Functional Programming 2013)

- **Implementation strategies**
  - See above

- **Semantics** what should happen when programs are run
  - TransCCS\textsuperscript{m}: bisimulation-based theory Fossacs 2014
Co-operating Transactions: Issues

- **Language Design**
  - **Transaction Synchronisers** (Luchangco et al 2005)
  - **cJoin with commits** Bruni, Melgratti, Montanari ENTCS 2004
  - **Transactional Events** for ML (Fluet, Grossman et al. ICFP 2008)
  - **Communication Memory Transactions** (Lesani, Palsberg PPoPP 2011)
  - **... Abstractions for Concurrent Consensus** (Spaccasassi, Koutavas, Trends in Functional Programming 2013)
- **Implementation strategies**
  - See above
- **Semantics** what should happen when programs are run
  - **TransCCS**: Testing-based semantic theory (Concur 2010, Aplas 2010)
  - **TransCCS\textsuperscript{m}**: bisimulation-based theory Fossacs 2014
Communicating Memory Transactions  
Lesani Palsberg

- Builds on optimistic semantics of memory transactions by 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
**TCCSm**

An extension of CCS with communicating transactions.

1. **Simple language:** 3 additional language constructs
2. Reduction semantics based on merging of mutually dependent transactions
3. **Intricate concurrent and transactional behaviour:**
   - encodes restarting, and non-restarting transactions
   - does not limit communication between transactions
   - no nesting of transactions for simplicity
4. **Behavioural theory:** based on standard contextual equivalence
   - reduction barbed congruence
5. **History based bisimulations** which are fully abstract for behavioural theory
TCCS$^m$

Syntax: $P, Q ::= \sum \mu_i.P_i$ guarded choice
$\mid P \parallel Q$ parallel
$\mid \nu a.P$ hiding
$\mid \text{rec}X.P$ recursion
$\mid [P \triangleright_k Q]$ running transaction named $k$
$\mid \text{co}$ commit
$\mid [P \triangleright Q]$ uninitiated transaction

Transaction $[P \triangleright_k Q]$
- execute $P$ to completion (co)
- subject to random aborts
- if aborted, roll back all effects of $P$ and initiate $Q$
- roll back includes ... environmental impact of $P$
TransCCS\textsuperscript{m}

Syntax: \allowbreak P, Q \allowbreak ::= \sum \mu_i.P_i \quad \text{guarded choice} \\
\quad | \quad P \mid Q \quad \text{parallel} \\
\quad | \quad \nu a.P \quad \text{hiding} \\
\quad | \quad \text{rec} X.P \quad \text{recursion} \\
\quad | \quad [P \triangleright_k Q] \quad \text{running transaction named } k \\
\quad | \quad \text{co} \quad \text{commit} \\
\quad | \quad [P \triangleright Q] \quad \text{uninitiated transaction} \\

Transaction \allowbreak [P \triangleright_k Q] \\
\quad \triangleright \quad \text{execute } P \text{ to completion (co)} \\
\quad \triangleright \quad \text{subject to random aborts} \\
\quad \triangleright \quad \text{if aborted, roll back all effects of } P \text{ and initiate } Q \\
\quad \triangleright \quad \text{roll back includes ... environmental impact of } P
**TCCS^m**

Syntax: \( P, Q ::= \sum \mu_i.P_i \) guarded choice  
\( P \mid Q \) parallel  
\( \nu a.P \) hiding  
\( \text{rec} X.P \) recursion  
\( [P \triangleright_k Q] \) running transaction named \( k \)  
\( \text{co} \) commit  
\( [P \triangleright Q] \) uninitiated transaction

Transaction \( [P \triangleright_k Q] \)

- execute \( P \) to completion (co)
- subject to random aborts
- if aborted, roll back all effects of \( P \) and initiate \( Q \)
- roll back includes ... environmental impact of \( P \)
Co-operating Transactions

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

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

where

\[
T_a = [\bar{d}.\bar{b}.(\text{co } | \ a) \triangleright_{k_1} 0]
\]
\[
T_b = [\bar{c}.(\text{co } | \ b) \triangleright_{k_2} 0]
\]
\[
T_c = [\bar{e}.c.\text{co} \triangleright_{k_3} 0]
\]
\[
P_d = d.R_d
\]
\[
P_e = e.R_e
\]

- if \( T_c \) aborts, what roll-backs are necessary?
- When can action \( a \) be considered permanent?
- When can code \( P_d \) be considered permanent?
Rollbacks and Commits

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

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

where

\[
T_a = \left[ \overline{d.b.(co \mid a)} \triangleright_{k_1} \emptyset \right]
\]
\[
T_b = \left[ \overline{c.(co \mid b)} \triangleright_{k_2} \emptyset \right]
\]
\[
T_c = \left[ \overline{e.c.co} \triangleright_{k_3} \emptyset \right]
\]
\[
P_d = d.R_d
\]
\[
P_e = e.R_e
\]

▷ if \( T_c \) aborts, what roll-backs are necessary?
▷ When can action \( a \) be considered permanent?
▷ When can code \( P_d \) be considered permanent?
Rollbacks and Commits

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

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

where

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

- if \( T_c \) aborts, what roll-backs are necessary?
- When can action \( a \) be considered permanent?
- When can code \( P_d \) be considered permanent?
Tentative vs Permanent actions for bisimulations

\[
\left[ a.b.co.P + a.c.\emptyset \triangleright_k \emptyset \right] \xrightarrow{k(a)} \text{tentative } a
\]

\[
\xrightarrow{k(b)} \text{tentative } b
\]
Tentative vs Permanent actions for bisimulations

\[[a.b.co.P + a.c.\emptyset \triangleright_k \emptyset] \xrightarrow{a} \text{permanent } a\]

\[[a.b.co.P + a.c.\emptyset \triangleright_k \emptyset] \xrightarrow{b} \text{permanent } b\]

\[[a.b.co.P + a.c.\emptyset \triangleright_k \emptyset] \xrightarrow{cok} \text{commit } k\]

\[[a.b.co.P + a.c.\emptyset \triangleright_k \emptyset] \xrightarrow{k(a)} \text{tentative } a\]

\[[a.b.co.P + a.c.\emptyset \triangleright_k \emptyset] \xrightarrow{k(c)} \text{tentative } c\]
Tentative vs Permanent actions for bisimulations

\[ [a.b.co.P + a.c.0 \triangleright_k 0] \xrightarrow{a} \text{permanent } a \]

\[ b \xrightarrow{} \text{permanent } b \]

\[ \text{co} k \xrightarrow{} \text{commit } k \]

\[ [a.b.co.P + a.c.0 \triangleright_k 0] \xrightarrow{k(a)} \text{tentative } a \]

\[ k(c) \xrightarrow{} \text{tentative } c \]
Remembering via Histories: $H \rightharpoonup P$

\[ \varepsilon \rightharpoonup [a.b.co.P + a.c.\emptyset \rightharpoonup_k \emptyset] \xrightarrow{k_1} k_1(a) \rightharpoonup [b.co.P \rightharpoonup_k \emptyset] \]

\[ \xrightarrow{k_2} k_2(a).k_2(b) \rightharpoonup [co.P \rightharpoonup_k \emptyset] \]

\[ \xrightarrow{co} a.b \rightharpoonup P \] permanent $a, b$

Configurations: $H \rightharpoonup P$

where $H$ remembers

- tentative actions what commits they depend on
- aborted actions
- permanent actions
Remembering via Histories: $H \triangleright P$

\[
\varepsilon \triangleright [a.b.co.P + a.c.\emptyset \triangleright_k \emptyset] \xrightarrow{k_1} k_1(a) \triangleright [b.co.P \triangleright_{k_1} \emptyset]
\]

\[
\xrightarrow{k_2} k_2(a). k_2(b) \triangleright [co.P \triangleright_{k_2} \emptyset]
\]

\[
\xrightarrow{co} a. b \triangleright P \quad \text{permanent} \ a, b
\]

Configurations: $H \triangleright P$

where $H$ remembers

- tentative actions what commits they depend on
- aborted actions
- permanent actions
Bisimulations

\[ H_1 \triangleright P_1 \approx_{\text{bisim}} H_2 \triangleright P_2 \]

whenever

- \( H_1, H_2 \) are consistent permanent actions agree

- \( H_1 \triangleright P_1 \xrightarrow{\lambda} H'_1 \triangleright P'_1 \) implies \( H_2 \triangleright P_2 \xrightarrow{\lambda} H'_2 \triangleright P'_2 \) such that
  \[ H'_1 \triangleright P_1 \approx_{\text{bisim}} H'_2 \triangleright P'_2 \]

- ...  

Intricacies:

- Commits/aborts treated as internal actions
- Dummy actions allowed
Bisimulations

\[ H_1 \triangleright P_1 \approx_{\text{bisim}} H_2 \triangleright P_2 \]

whenever

- \( H_1, H_2 \) are consistent permanent actions agree

- \( H_1 \triangleright P_1 \xrightarrow{\lambda} H'_1 \triangleright P'_1 \) implies \( H_2 \triangleright P_2 \xrightarrow{\lambda} H'_2 \triangleright P'_2 \) such that
  \[ H'_1 \triangleright P_1 \approx_{\text{bisim}} H'_2 \triangleright P'_2 \]

- \( \ldots \)

Intricacies:

- Commits/aborts treated as internal actions
- Dummy actions allowed
Examples

\[
\llbracket a.b.co + a.c.\emptyset \rhd_k \emptyset \rrbracket \ \approx_{\text{bisim}} \ \llbracket a.b.co \rhd_k \emptyset \rrbracket
\]

\[
\llbracket a.b.co + a.c.co \rhd_k \emptyset \rrbracket \ \approx_{\text{bisim}} \ \llbracket a.(b.co + c.co) \rhd_k \emptyset \rrbracket
\]

\[
\text{rec}X. \llbracket \tau.b.co + \tau.c.co \rhd_k X \rrbracket \ \approx_{\text{bisim}} \ \text{rec}X. \llbracket \tau.(b.co + c.co) \rhd_k X \rrbracket
\]

\[
R \mid \text{rec}X. \llbracket a.b.\emptyset \rhd_k X \rrbracket \ \approx_{\text{bisim}} \ R
\]

\[
P \ \approx_{\text{bisim}} \ Q
\]

where

\[
P = \nu p. \ [a.p.co.R \rhd_k \emptyset] \ | \ [b.p.co.S \rhd_l \emptyset]
\]

\[
Q = \llbracket a.b.co.(R \mid S) + b.a.co.(R \mid S) \rhd_m \emptyset \rrbracket
\]
Full-abstraction

Thm:

For $TCCS^m$, 

$$\varepsilon \triangleright P \approx_{bisim} \varepsilon \triangleright Q \iff P \approx_{cxt} Q$$

where $\approx_{cxt}$ is a standard contextual equivalence
Full-abstraction

Thm:

For $TCCS^m$, 

$$
\varepsilon \triangleright P \approx_{bisim} \varepsilon \triangleright Q \iff P \approx_{cxt} Q
$$

where $\approx_{cxt}$ is a standard contextual equivalence

THANKS