Co-operating/Communicating Transactions

a survey

Matthew Hennessy

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

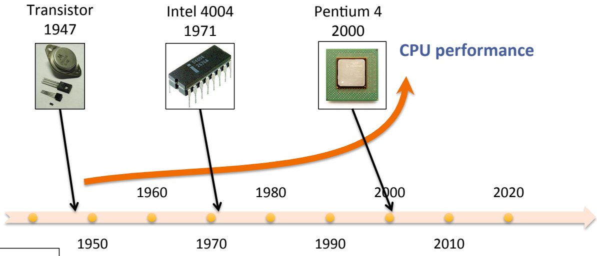
Bertinoro, June 2014

TRINITY COLLEGE DUBLIN
COLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH

science foundation ireland

Outline

First software crisis




```
01101…
```

```
MOV AL,61h
ASR AL
…
```

Software Crisis

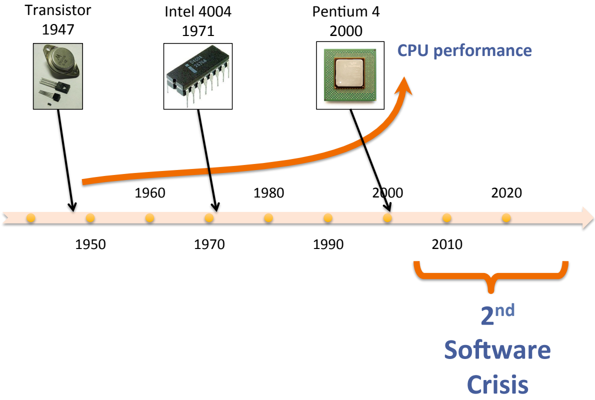
Explosion in high-level programming languages

Precursors:
Fortran, Algol, Lisp

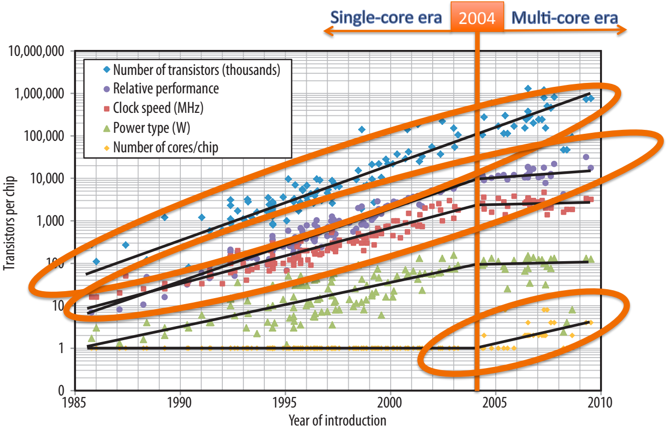
- Procedural (C, Pascal, Ada, …)
- Object-Oriented (C++, Java, Eiffel, …)
- Functional (ML, Scheme, Haskell, …)
- Declarative (Prolog, SQL, …)

New **Building blocks** for software

Second software crisis



Second software crisis reasons



[“The Future of Computing Performance: Game Over or Next Level?”
US National Research Council Report 2011]

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

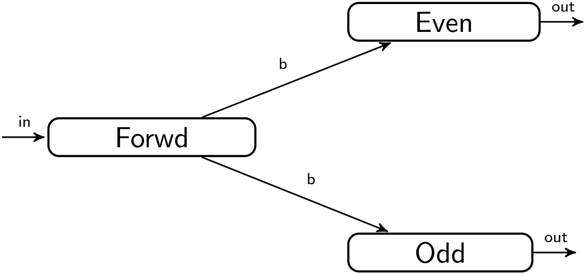
Standard Transactions on which STM is based

- 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

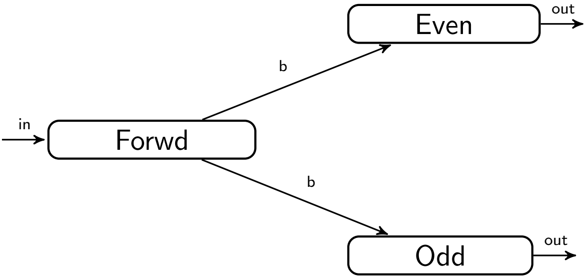


$\text{Forwd} \Leftarrow \text{in}?(x)\,.b!\langle x\rangle\,.\text{Forwd}$

$\text{Even} \Leftarrow \text{atomic}[\![b?(x)\,.\texttt{if}\ \text{even}(x)\ \texttt{then}\ \text{out}!\langle f(x)\rangle\,.(\mathit{commit} \mid \text{Even})$
$\texttt{else}\ \text{abrt\&retry}]\!]$

$\text{Odd} \Leftarrow \text{atomic}[\![b?(x)\,.\texttt{if}\ \text{odd}(x)\ \texttt{then}\ \text{out}!\langle g(x)\rangle\,.(\mathit{commit} \mid \text{Odd})$
$\texttt{else}\ \text{abrt\&retry}]\!]$

An example



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

$\text{Even} \Leftarrow \text{atomic}[\![b?(x)\,.\texttt{if}\ \text{even}(x)\ \texttt{then}\ \text{out}!\langle f(x)\rangle\,.(\mathit{commit} \mid \text{Even})$
$\qquad \texttt{else}\ \text{abrt\&retry}]\!]$

$\text{Odd} \Leftarrow \text{atomic}[\![b?(x)\,.\texttt{if}\ \text{odd}(x)\ \texttt{then}\ \text{out}!\langle g(x)\rangle\,.(\mathit{commit} \mid \text{Odd})$
$\qquad \texttt{else}\ \text{abrt\&retry}]\!]$

Example: three-way rendezvous

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

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

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

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

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

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

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

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

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

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

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^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^m: bisimulation-based theory Fossacs 2014

Communicating 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

$TCCS^m$

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
		$\mid$	$P \mid Q$	parallel
		$\mid$	$\nu a.P$	hiding
		$\mid$	$\mathtt{rec}X.P$	recursion
		$\mid$	$[\![P \triangleright_k Q]\!]$	running transaction named k
		$\mid$	co	commit
		$\mid$	$[\![P \blacktriangleright Q]\!]$	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

$TCCS^m$

Syntax:

$$
\begin{array}{lcll}
P, Q & ::= & \sum \mu_i.P_i & \text{guarded choice} \\
 & \mid & P \mid Q & \text{parallel} \\
 & \mid & \nu a.P & \text{hiding} \\
 & \mid & \mathtt{rec}X.P & \text{recursion} \\
 & \mid & [\![P \triangleright_k Q]\!] & \text{running transaction named } k \\
 & \mid & \mathtt{co} & \text{commit} \\
 & \mid & [\![P \blacktriangleright Q]\!] & \text{uninitiated transaction}
\end{array}
$$

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

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

$TCCS^m$

Syntax:

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

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

- execute P to completion ($\texttt{co}$)
- 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

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

- 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: $\boxed{a \leftarrow \text{needs co-operation of} \rightarrow \overline{a}}$

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

where

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

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

Rollbacks and Commits

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

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

where

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

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

Tentative vs Permanent actions

for bisimulations

$$[\![a.b.\text{co}.P + a.c.\mathbb{0} \rhd_k \mathbb{0}]\!] \xrightarrow{k(a)} \quad \text{tentative } a$$

$$\xrightarrow{k(b)} \quad \text{tentative } b$$

Tentative vs Permanent actions

for bisimulations

$[\![a.b.\mathrm{co}.P + a.c.\mathbb{0} \triangleright_k \mathbb{0}]\!]$	$\xrightarrow{a}$	permanent a
	$\xrightarrow{b}$	permanent b
	$\xrightarrow{\mathrm{co}\,k}$	commit k
$[\![a.b.\mathrm{co}.P + a.c.\mathbb{0} \triangleright_k \mathbb{0}]\!]$	$\xrightarrow{k(a)}$	tentative a
	$\xrightarrow{k(c)}$	tentative c

Tentative vs Permanent actions

for bisimulations

$$[\![a.b.\mathrm{co}.P + a.c.\mathbb{0} \rhd_k \mathbb{0}]\!] \xrightarrow{a} \quad \text{permanent } a$$

$$\xrightarrow{b} \quad \text{permanent } b$$

$$\xrightarrow{\mathrm{co}k} \quad \text{commit } k$$

$$[\![a.b.\mathrm{co}.P + a.c.\mathbb{0} \rhd_k \mathbb{0}]\!] \xrightarrow{k(a)} \quad \text{tentative } a$$

$$\xrightarrow{k(c)} \quad \text{tentative } c$$

Remembering via Histories: $H \rhd P$

$$\varepsilon \rhd [\![a.b.co.P + a.c.\mathbb{0} \rhd_k \mathbb{0}]\!] \xrightarrow{k_1} k_1(a) \rhd [\![b.co.P \rhd_{k_1} \mathbb{0}]\!]$$ tentative

$$\xrightarrow{k_2} k_2(a).\, k_2(b) \rhd [\![co.P \rhd_{k_2} \mathbb{0}]\!]$$ tentative

$$\xrightarrow{co} a.\, b \rhd P$$ permanent a, b

Configurations: $H \rhd P$

where H remembers

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

Remembering via Histories: $H \triangleright P$

$$\varepsilon \triangleright [\![a.b.\mathsf{co}.P + a.c.\mathbb{0} \triangleright_k \mathbb{0}]\!] \xrightarrow{k_1} k_1(a) \triangleright [\![b.\mathsf{co}.P \triangleright_{k_1} \mathbb{0}]\!] \quad \text{tentative}$$

$$\xrightarrow{k_2} k_2(a).\, k_2(b) \triangleright [\![\mathsf{co}.P \triangleright_{k_2} \mathbb{0}]\!] \quad \text{tentative}$$

$$\xrightarrow{\mathsf{co}} a.\, b \triangleright P \quad \text{permanent } a, b$$

Configurations: $H \triangleright P$

where H remembers

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

Bisimulations

$$H_1 \rhd P_1 \approx_{\text{bisim}} H_2 \rhd P_2$$

whenever

- H_1, H_2 are consistent permanent actions agree
- $H_1 \rhd P_1 \xrightarrow{\lambda} H_1' \rhd P_1'$ implies $H_2 \rhd P_2 \stackrel{\lambda}{\Rightarrow} H_2' \rhd P_2'$ such that $H_1' \rhd P_1 \approx_{\text{bisim}} H_2' \rhd P_2'$
- ...

Intricacies:

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

Bisimulations

$$H_1 \rhd P_1 \;\approx_{\text{bisim}}\; H_2 \rhd P_2$$

whenever

- H_1, H_2 are consistent permanent actions agree
- $H_1 \rhd P_1 \xrightarrow{\lambda} H_1' \rhd P_1'$ implies $H_2 \rhd P_2 \stackrel{\lambda}{\Rightarrow} H_2' \rhd P_2'$ such that $H_1' \rhd P_1 \;\approx_{\text{bisim}}\; H_2' \rhd P_2'$
- ...

Intricacies:

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

Examples

$$[\![a.b.\mathrm{co} + a.c.\mathbb{0} \triangleright_k \mathbb{0}]\!] \overset{?}{\approx}_{\mathrm{bisim}} [\![a.b.\mathrm{co} \triangleright_k \mathbb{0}]\!]$$

$$[\![a.b.\mathrm{co} + a.c.\mathrm{co} \triangleright_k \mathbb{0}]\!] \overset{?}{\approx}_{\mathrm{bisim}} [\![a.(b.\mathrm{co} \; + \; c.\mathrm{co}) \triangleright_k \mathbb{0}]\!]$$

$$\mathtt{rec}X.\, [\![\tau.b.\mathrm{co} + \tau.c.\mathrm{co} \triangleright_k X]\!] \overset{?}{\approx}_{\mathrm{bisim}} \mathtt{rec}X.\, [\![\tau.(b.\mathrm{co} + c.\mathrm{co}) \triangleright_k X]\!]$$

$$R \mid \mathtt{rec}X.\, [\![a.b.\mathbb{0} \triangleright_k X]\!] \overset{?}{\approx}_{\mathrm{bisim}} R$$

$$P \overset{?}{\approx}_{\mathrm{bisim}} Q$$

where
$$P = \nu p.\; [\![a.p.\mathrm{co}.R \triangleright_k \mathbb{0}]\!] \mid [\![b.\overline{p}.\mathrm{co}.S \triangleright_l \mathbb{0}]\!]$$
$$Q = [\![a.b.\mathrm{co}.(R \mid S) + b.a.\mathrm{co}.(R \mid S) \triangleright_m \mathbb{0}]\!]$$

Full-abstraction

Thm:

For $TCCS^m$,

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

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

THANKS

Full-abstraction

Thm:

For $TCCS^m$,

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

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

THANKS