Towards reversible languages and systems

Jean-Bernard Stefani

joint work with I. Lanese, M. Lienhardt, C.A. Mezzina and A. Schmitt

INRIA, France

From JSR 220 v3.0 “EJB Core Contracts and Requirements” (May 2006), pp 353:

> *The Bean Provider and Application Assembler must avoid creating applications that would result in inconsistent caching of data in the same transaction by multiple session objects.*

Cf nested transactions [Moss 85], multithreaded transactions [Kienzle 01]

From JSR 220 v3.0 "EJB Core Contracts and Requirements", pp 357:

> *The Bean Provider must do one of the following to ensure data integrity before throwing an application exception from an enterprise bean instance:*
>
> - *Ensure that the instance is in a state such that a clients attempt to continue and/or commit the transaction does not result in loss of data integrity. For example, the instance throws an application exception indicating that the value of an input parameter was invalid before the instance performed any database updates.*
> - *If the application exception is not specified to cause transaction rollback, mark the transaction for rollback using the EJBContext.setRollbackOnly method before throwing the application exception. Marking the transaction for rollback will ensure that the transaction can never commit.*

Cf atomic exception handling [Fetzer07], failboxes [Jacobs 09]

We do not understand if/how the following compose

Exception handling
Concurrency
Transactions
Compensations
Groups, Replicas
Checkpoints, Rollback
Objects, Actors, Components

Enlarging/formalizing the bag of tricks

- Exception handling [Collet 07][Fetzer 07]
- Failboxes [Jacobs 09]
- Transactors [Field 05]
- Transactional Events [Fluet 06][Grossman 08]
- Transactional memory [Grossman 08][Abadi10]
- Transactional calculi [Acciai 07][deVries 10]
- Compensations [Bruni 05][Lanese10]
- Logs and conclaves [Chothia 04]
- Stabilizers and checkpoints [Ziarek10]

Wanted

Semantical and Operational Foundations for Fault-Tolerant Programming

Roadmap

Roadmap

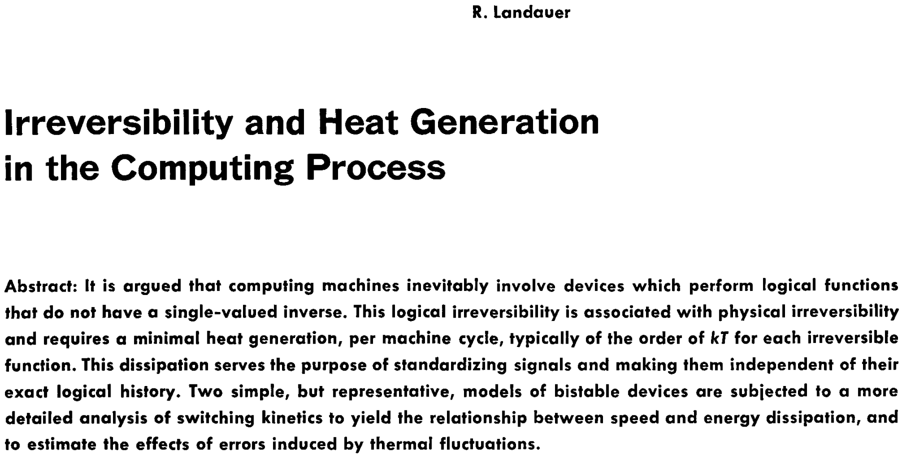
Undo in fault-tolerant constructs

- Application undo
- System undo (ROC)
- Checkpoint/Rollback schemes
- Transaction rollback

What if we could undo every action ?

Language support for Recovery Oriented Computing (ROC)

Reversibility



R. Landauer

Irreversibility and Heat Generation in the Computing Process

Abstract: It is argued that computing machines inevitably involve devices which perform logical functions that do not have a single-valued inverse. This logical irreversibility is associated with physical irreversibility and requires a minimal heat generation, per machine cycle, typically of the order of *kT* for each irreversible function. This dissipation serves the purpose of standardizing signals and making them independent of their exact logical history. Two simple, but representative, models of bistable devices are subjected to a more detailed analysis of switching kinetics to yield the relationship between speed and energy dissipation, and to estimate the effects of errors induced by thermal fluctuations.

Using reversibility

Enabling defeasible partial agreements

- Systematic rollback-recovery
- Operational primitive for transaction models
  - all or nothing execution
  - complement with isolation, compensation

Using reversibility

Enabling causality tracking

- Debugging
- Diagnosis
- Simulation
- Fault isolation

Work programme

- Reversible distributed programming model
  - reversible substrate, controlling reversibility
  - primitives, operational semantics, behavioral theory
- Composable FT abstractions
  - benchmarks: transaction, checkpoint/rollback, exception handling schemes
  - combination with modularity features
- Prototype
  - implementation costs / tradeoffs for distributed reversibility
  - language experiments

Roadmap

Expected features

Higher-order π-calculus substrate

- Concurrency, functional and imperative features in a simple setting
- Formal (small-step) operational semantics
- Behavioral theory (program equivalence)

Expected features

Reversible actions

If program M can evolve into M', then M' can evolve back into M:

$$M \;\rightarrow\; M' \text{ implies } M' \;\rightarrow\; M$$

Expected features

Causal consistency

Any program state reached by reversal could have been reached during the past computation (just by exchanging the order of concurrent actions)

Determinism in a distributed setting

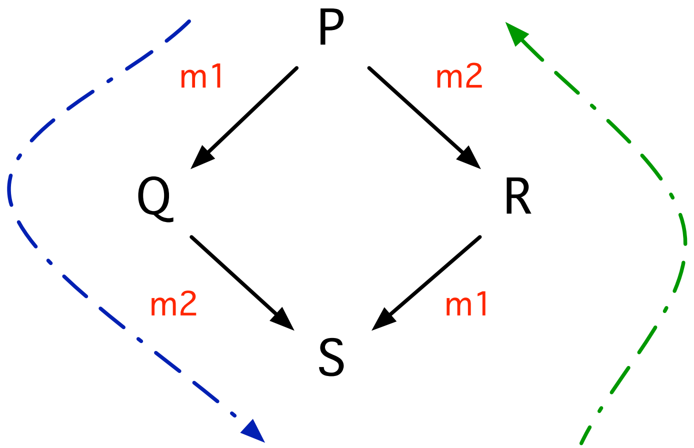
Causal Consistency

$$\begin{tikzcd}
& P \arrow[dl, "m1"'] \arrow[dr, "m2"] & \\
Q \arrow[dr, "m2"'] & & R \arrow[dl, "m1"] \\
& S &
\end{tikzcd}$$

Causal Consistency

$$\begin{tikzcd}
 & P \arrow[dl, "m1"'] \arrow[dr, "m2"] & \\
Q \arrow[dr, "m2"'] & & R \arrow[dl, "m1"] \\
 & S &
\end{tikzcd}$$

Causal Consistency


$$\begin{tikzcd}
 & P \arrow[dl, "m1"'] \arrow[dr, "m2"] & \\
Q \arrow[dr, "m2"'] & & R \arrow[dl, "m1"] \\
 & S &
\end{tikzcd}$$

Higher-order π-calculus

Syntax

$$
\begin{array}{rll}
P, Q ::= & \mathbf{0} & \textit{null process} \\
\mid & X & \textit{variable} \\
\mid & \nu a.\, P & \textit{new name} \\
\mid & (P \mid Q) & \textit{parallel composition} \\
\mid & a\langle P \rangle & \textit{message} \\
\mid & (a(X) \triangleright P) & \textit{trigger}
\end{array}
$$

$a \in \mathcal{N}$

Higher-order π-calculus

Operational semantics

Defined by means of a binary relation $\rightarrow$ between programs: $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$

$P \rightarrow Q$ reads "P reduces (or evolves into) Q"

Higher-order π-calculus

Reduction rules

$$a\langle Q\rangle \mid (a(X) \triangleright P) \rightarrow P\{^{Q}/_{X}\} \qquad \frac{P \rightarrow Q}{\mathbb{E}[P] \rightarrow \mathbb{E}[Q]}$$

$$\mathbb{E} ::= \bullet \mid \nu a.\,\mathbb{E} \mid (P \mid \mathbb{E}) \mid (\mathbb{E} \mid P) \qquad \textit{evaluation context}$$

HOπ examples

$$a\langle P\rangle \mid (a(X) \triangleright \mathbf{0}) \;\rightarrow\; \mathbf{0}$$

$$a\langle P\rangle \mid (a(X) \triangleright X \mid X) \;\rightarrow\; P \mid P$$

$$Q \mid a\langle P\rangle \mid (a(X) \triangleright \mathbf{0}) \;\rightarrow\; Q \mid \mathbf{0}$$

Higher-order π-calculus

Syntactical equivalence

Given by relation $\equiv \subseteq \mathcal{P} \times \mathcal{P}$

$\equiv$ is the smallest congruence that obeys the rules:

$$A \mid B \equiv B \mid A \qquad A \mid (B \mid C) \equiv (A \mid B) \mid C \qquad A \mid \mathbf{0} \equiv A \qquad \nu a.\, \mathbf{0} \equiv \mathbf{0}$$

$$\nu a.\, \nu b.\, A \equiv \nu b.\, \nu a.\, A \qquad (\nu a.\, A) \mid B \equiv \nu a.\, (A \mid B)$$

HOπ examples

$$Q \mid a\langle P\rangle \mid (a(X) \triangleright \mathbf{0}) \;\rightarrow\; Q$$

$$a\langle P\rangle \mid Q \mid (a(X) \triangleright X \mid X) \;\rightarrow\; P \mid Q \mid P$$

HOπ examples

$$\texttt{B} = (a(X) \triangleright X \mid a\langle X\rangle)$$
$$\texttt{Bang}(P) = \nu a.\, a\langle P \mid \texttt{B}\rangle \mid \texttt{B}$$
$$P = c(Y) \triangleright Y \mid Y$$

$$c\langle Q\rangle \;\mid\; \texttt{Bang}(P) \;\rightarrow\; c\langle Q\rangle \mid P \mid \texttt{Bang}(P) \;\rightarrow\; Q \mid Q \mid \texttt{Bang}(P)$$

Higher-order π-calculus

Observables

An event on channel a is observable from process P, notation $P{\downarrow_a}$ if

$$P \equiv \nu\vec{e}.\, a\langle Q\rangle \mid R \quad \text{and} \quad a \notin \vec{e}$$

Semantical equivalence

Two programs P, Q are (weakly) barbed bisimilar, notation $P \approx Q$, if the following conditions hold

- if $P{\downarrow_a}$, then $Q \rightarrow^* Q'$ and $Q'{\downarrow_a}$
- if $P \rightarrow P'$, then $Q \rightarrow^* Q'$ and $P' \approx Q'$
- and the converse for Q and P.

$\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$

HOπ examples

$$P \equiv Q \quad \text{implies} \quad P \approx Q$$

$$\nu a.\, a\langle P \rangle \;\approx\; \mathbf{0}$$

$$\nu a.\, a\langle P \rangle \mid (a(X) \triangleright \mathbf{0}) \;\approx\; \mathbf{0}$$

A reversible HOπ: rho-π

To make HOπ reversible:

- Log each action (message receipt)
- Uniquely identify action participants (thread tags)

A reversible HOπ: rho-π

Syntax

$$P, Q ::= \mathbf{0} \mid X \mid \nu a.\, P \mid (P \mid Q) \mid a\langle P\rangle \mid (a(X) \triangleright P)$$

$M, N ::=$		*configurations*
	$\mathbf{0}$	*null configuration*
	$\mid \nu u.\, M$	*restriction*
	$\mid (M \mid N)$	*parallel*
	$\mid \kappa : P$	*thread*
	$\mid [m; k]$	*memory*

$$a \in \mathcal{N},\ k \in \mathcal{T},\ u \in \mathcal{N} \cup \mathcal{T}$$

A reversible HOπ: rho-π

Intuitions:

- $\kappa : P$ thread of computation (process) P uniquely identified by tag κ
- $[m; k]$ log of occurrence κ of action m (message receipt)

A reversible HOπ: rho-π

Syntax

$$\kappa ::= k \mid \langle h, \tilde{h} \rangle \cdot k \qquad \textit{tags}$$

$$m ::= ((\kappa_1 : a\langle P \rangle) \mid (\kappa_2 : a(X) \triangleright Q)) \qquad \textit{action record}$$

A reversible HOπ: rho-π

Operational semantics

Defined by means of a binary relation $\rightarrow \subseteq \mathcal{P} \times \mathcal{P}$, which is the union of a forward reduction relation $\twoheadrightarrow$ and of a backward one $\rightsquigarrow$.

A reversible HOπ: rho-π

Reduction rules

$$\frac{m = (\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q)}{(\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright Q) \twoheadrightarrow \nu k.\,(k : Q\{^P/_X\}) \mid [m; k]}$$

$$(k : P) \mid [m; k] \rightsquigarrow m$$

rho-π examples

$$k_1 : a\langle P\rangle \mid k_2 : a(X) \triangleright \mathbf{0} \;\rightarrow\; \nu k.\, k : \mathbf{0} \mid [k_1 : a\langle P\rangle \mid k_2 : a(X) \triangleright \mathbf{0}; k]$$

$$k : R \mid [k_1 : a\langle Q\rangle \mid k_2 : a(X) \triangleright P \;;\; k] \rightarrow k_1 : a\langle Q\rangle \mid k_2 : a(X) \triangleright P$$

A reversible HOπ: rho-π

Syntactical equivalence: building unique tags

$$\kappa : \nu a.\, P \equiv \nu a.\, \kappa : P$$

$$k : \prod_{i=1}^{n} \tau_i \equiv \nu \tilde{h}.\, \prod_{i=1}^{n} (\langle h_i, \tilde{h} \rangle \cdot k : \tau_i) \quad \tilde{h} = \{h_1, \dots, h_n\}$$

τ_i are non-null threads (ie messages or triggers)

Results

Theorem (Loop Lemma)

$$M \to^* N \text{ if and only if } N \to^* M$$

Theorem (Causal consistency)

Let σ_1 and σ_2 be two coinitial traces. Then $\sigma_1 \asymp \sigma_2$ if and only if σ_1 and σ_2 are cofinal.

Theorem (Faithful encoding)

There exists a compositional encoding $(\!|\bullet|\!) : \mathcal{P}_{\textit{rho-}\pi} \to \mathcal{P}_{HO\pi}$ such that for all configurations M:

$$M \approx (\!|M|\!)$$

What we have obtained

A small formal model for reversible concurrent computation

“Ballistic” programs that non-deterministically explore their state space

What we have obtained

- A model with fine-grained logs for action reversal
- A model with explicit causal information on computations
  - Cf causal logging in distributed checkpoint/rollback schemes
- A form of causal semantics for HOπ

What is missing ?

- Controlling reversibility
- Dealing with irreversible actions
    - eg, what about I/O ?

Roadmap

The roll-π calculus

- Asynchronous, higher-order π
  - only forward reduction: receiving a process on a channel
- Explicit rollback instruction
  - backward reduction triggered by roll

roll-π reversibility/causality substrate

- Tagged processes: $\kappa : P$
- Message receipts create memories : $[m; k]$
  - with $m = \kappa_1 : a\langle P \rangle \mid \kappa_2 : a(X) \triangleright_\gamma Q$
- Unicity of tags ensured by structural congruence law :

$$k : \prod_{i=1}^{n} \tau_i \equiv \nu \tilde{h}.\ \prod_{i=1}^{n} \langle h_i, \tilde{h} \rangle \cdot k : \tau_i$$

$$\text{with}\ \ \tilde{h} = \{h_1, \ldots, h_n\} \quad n > 1$$

The roll-π calculus

Syntax

$$P, Q ::= \mathbf{0} \mid \ldots \mid a(X) \triangleright_{\gamma} P \mid \text{roll } k \mid \text{roll } \gamma$$

$$M, N ::= \mathbf{0} \mid \nu u.\, M \mid (M \mid N) \mid \kappa : P \mid [m; k]$$

$$\kappa ::= k \mid \langle h, \tilde{h} \rangle \cdot k \qquad m ::= (\kappa_1 : a\langle P \rangle) \mid (\kappa_2 : a(X) \triangleright_{\gamma} Q)$$

$$a, b, c \in \mathcal{N} \quad X \in \mathcal{V}_{\mathcal{P}} \quad \gamma \in \mathcal{V}_{\mathcal{K}} \quad u \in \mathcal{N} \cup \mathcal{K} \quad h, k \in \mathcal{K}$$

The roll-π calculus

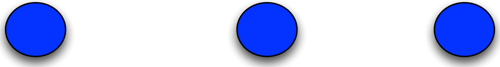
Reduction rules

$$\frac{m = (\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright_\gamma Q_2)}{(\kappa_1 : a\langle P\rangle) \mid (\kappa_2 : a(X) \triangleright_\gamma Q_2) \rightarrow \nu k.\, (k : Q_2\{{}^{P,k}/_{X,\gamma}\}) \mid [m; k]}$$

$$\frac{N \blacktriangleright k \qquad \texttt{complete}(N \mid [m;k] \mid (\kappa : \mathsf{roll}\ k))}{N \mid [m;k] \mid (\kappa : \mathsf{roll}\ k) \rightarrow m \mid N \lightning_k}$$

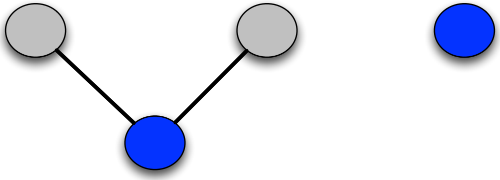
Rollback example

$$k_1 : a\langle 0\rangle \quad (k_2 : a(X) \triangleright_\gamma b\langle roll\ \gamma\rangle) \quad (k_3 : b(X) \triangleright c\langle 0\rangle | X)$$



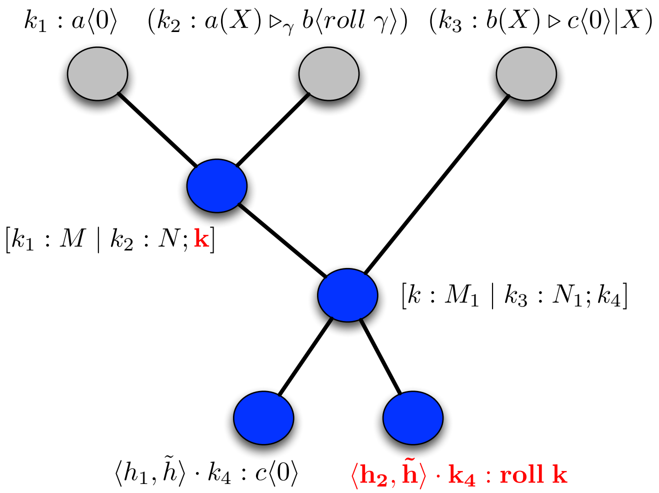
Rollback example

$k_1 : a\langle 0\rangle \quad (k_2 : a(X) \triangleright_\gamma b\langle roll\ \gamma\rangle) \quad (k_3 : b(X) \triangleright c\langle 0\rangle | X)$



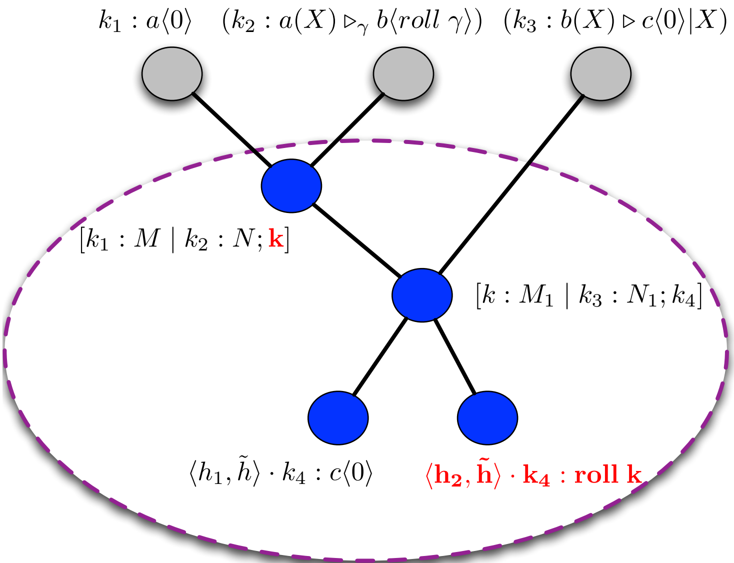
$[k_1 : M \mid k_2 : N; \mathbf{k}] \mid k : b\langle roll\ k\rangle$

Rollback example



$k_1 : a\langle 0\rangle$
$(k_2 : a(X) \triangleright_\gamma b\langle roll\ \gamma\rangle)$
$(k_3 : b(X) \triangleright c\langle 0\rangle | X)$
$[k_1 : M \mid k_2 : N; \mathbf{k}]$
$[k : M_1 \mid k_3 : N_1; k_4]$
$\langle h_1, \tilde{h}\rangle \cdot k_4 : c\langle 0\rangle$
$\langle \mathbf{h_2}, \tilde{\mathbf{h}}\rangle \cdot \mathbf{k_4} : \mathbf{roll\ k}$

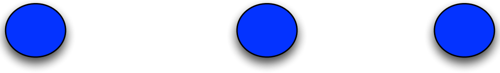
Rollback example



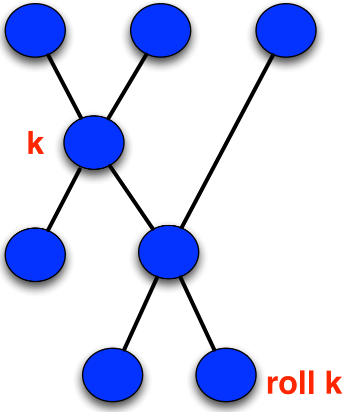
$k_1 : a\langle 0\rangle$
$(k_2 : a(X) \triangleright_\gamma b\langle roll\ \gamma\rangle)$
$(k_3 : b(X) \triangleright c\langle 0\rangle | X)$
$[k_1 : M \mid k_2 : N; \mathbf{k}]$
$[k : M_1 \mid k_3 : N_1; k_4]$
$\langle h_1, \tilde{h}\rangle \cdot k_4 : c\langle 0\rangle$
$\langle \mathbf{h_2}, \tilde{\mathbf{h}}\rangle \cdot \mathbf{k_4} : \mathbf{roll\ k}$

Rollback example

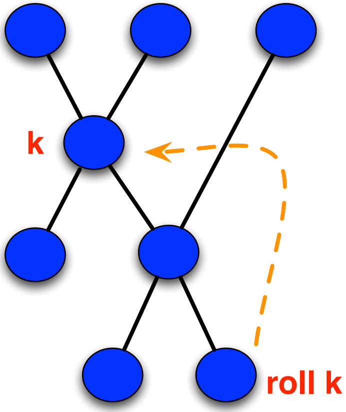
$$k_1 : a\langle 0\rangle \quad (k_2 : a(X) \triangleright_\gamma b\langle roll\ \gamma\rangle) \quad (k_3 : b(X) \triangleright c\langle 0\rangle | X)$$



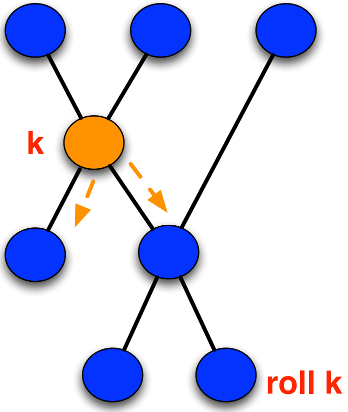
Concurrent non-atomic rollback:hint



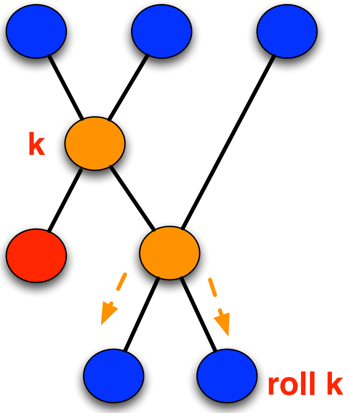
Concurrent non-atomic rollback:hint



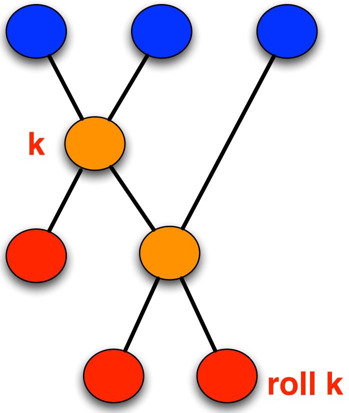
Concurrent non-atomic rollback:hint



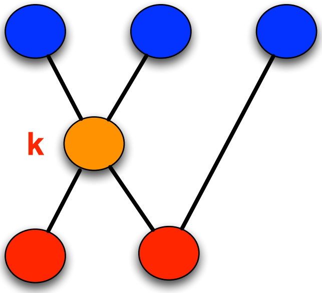
Concurrent non-atomic rollback:hint



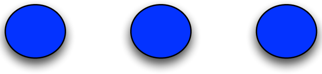
Concurrent non-atomic rollback:hint



Concurrent non-atomic rollback:hint



Concurrent non-atomic rollback:hint



Result

Theorem (Correspondence HL - LL)

For all configurations M:

$$M \ {}_{HL}{\approx}_{LL} \ M$$

Adding compensations to roll-π: croll-π

- New construct added: message with compensation: $a\langle P\rangle \div b\langle Q\rangle$
- Modified reduction rules:

$$\frac{m = (\kappa_1 : a\langle P\rangle \div C) \mid (\kappa_2 : a(X) \triangleright_\gamma Q_2)}{(\kappa_1 : a\langle P\rangle \div C) \mid (\kappa_2 : a(X) \triangleright_\gamma Q_2) \to \nu k.\, (k : Q_2\{^{P,k}/_{X,\gamma}\}) \mid [m;k]}$$

$$\frac{N \blacktriangleright k \qquad \texttt{complete}(N \mid [m;k] \mid (\kappa : \text{roll } k)) \qquad m' = \texttt{xtr}(m)}{N \mid [m;k] \mid (\kappa : \text{roll } k) \to m' \mid N\lightning_k}$$

$$\texttt{xtr}(a\langle P\rangle \div c\langle Q\rangle) = c\langle Q\rangle \qquad \texttt{xtr}(a\langle P\rangle) = a\langle P\rangle$$

Message compensations are *not* elementary

- Can trigger arbitrary compensation processes
- Can encode different compensation strategies

Endless retry

$$(\!| a\langle P\rangle \div \star |\!)_{er} = \nu t.\, Y \mid a\langle P\rangle \div t\langle Y\rangle$$

$$Y = t(Z) \triangleright Z \mid a\langle P\rangle \div t\langle Z\rangle$$

Bounded retries

$$(\!|a\langle P\rangle \div_{(n)} c\langle Q\rangle|\!)_{br} = \nu t.\,(t\langle Z\rangle \triangleright Z) \mid a\langle P\rangle \div t\langle \phi^{n}(c\langle Q\rangle)\rangle$$

$$\phi^{0}(c\langle Q\rangle) = c\langle Q\rangle$$

$$\phi^{n+1}(c\langle Q\rangle) = (t\langle Z\rangle \triangleright Z) \mid a\langle P\rangle \div t\langle \phi^{n}(c\langle Q\rangle)\rangle$$

Trigger compensations

$$(\!|a(X) \triangleright_\gamma P \div b\langle Q\rangle|\!)_{ct} = \nu c, d.\, \overline{c} \div \overline{d} \mid (c \triangleright_\gamma a(X) \triangleright P) \mid (d \triangleright b\langle Q\rangle)$$

Encoding transactions with croll-π

All-or-nothing processes can be encoded

- All-or-nothing: either complete execution or none at all
- Processes: no restriction on the form of transactional processes

Encoding transactions with croll-π

$$[P, Q] = \nu a, c.\, \overline{a} \div \overline{c} \mid (a \triangleright_\gamma \nu t.\, P \mid t\langle \text{roll } \gamma \rangle \mid (t(X) \triangleright X)) \mid (c \triangleright Q)$$

- *Committing* is consuming $t\langle \text{roll } \gamma \rangle$
- *Aborting* is releasing roll γ
- P executes till it commits or aborts
- On abort, the compensation process Q is released

Theorem

$[\bullet, \bullet]$ *is an all-or-nothing atomic construct with compensation*

Roadmap

Implementation questions

- How efficiently can we implement reversibility ?
- What are the time/space complexity lower bounds ?

A simple higher-order concurrent language: μOz

S	$::=$			Statements
			skip	Empty statement
		$\mid$	$S_1\ S_2$	Sequential composition
		$\mid$	**let** $x = v$ **in** S **end**	Variable declaration
		$\mid$	**if** x **then** S_1 **else** S_2 **end**	Conditional statements
		$\mid$	**thread** S **end**	Thread creation
		$\mid$	**let** $x = c$ **in** S **end**	Procedure declaration
		$\mid$	$\{\ x\ x_1 \ldots x_n\ \}$	Procedure call
		$\mid$	**let** $x =$ `NewPort` **in** S **end**	Port creation
		$\mid$	$\{$ `Send` $x\ y\ \}$	Send on a port
		$\mid$	**let** $x = \{$ `Receive` $y\ \}$ **in** S **end**	Receive from a port
v	$::=$		**true** $\mid$ **false**	Simple values
c	$::=$		**proc** $\{\ x_1 \ldots x_n\ \}$ S **end**	Procedure

μOz abstract machine

$$\frac{\langle \textbf{skip}\ T\rangle}{0} \;\Big\|\; \frac{T}{0}$$

$$\frac{\langle \textbf{let}\ x = v\ \textbf{in}\ S\ \textbf{end}\ T\rangle}{0} \;\Big\|\; \frac{\langle S\{^{x'}/_{x}\}\ T\rangle}{x' = v}\ \text{if } x' \text{ fresh}$$

$$\frac{\langle \textbf{let}\ x = c\ \textbf{in}\ S\ \textbf{end}\ T\rangle}{0} \;\Big\|\; \frac{\langle S\{^{x'}/_{x}\}\ T\rangle}{x' = \xi \parallel \xi : c}\ \text{if } x', \xi \text{ fresh}$$

$$\frac{\langle \textbf{if}\ x\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{end}\ T\rangle}{x = \textbf{true}} \;\Big\|\; \frac{\langle S_1\ T\rangle}{x = \textbf{true}}$$

$$\frac{\langle \textbf{if}\ x\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{end}\ T\rangle}{x = \textbf{false}} \;\Big\|\; \frac{\langle S_2\ T\rangle}{x = \textbf{false}}$$

μOz abstract machine

$$\frac{\langle \textbf{let}\ x = \texttt{NewPort}\ \textbf{in}\ S\ \textbf{end}\ T\rangle}{0} \;\Big\|\; \frac{\langle S\{{}^{x'}/_{x}\}\ T\rangle}{x' = \xi \parallel \xi : \bot} \quad \text{if } x', \xi \text{ fresh}$$

$$\frac{\langle \textbf{thread}\ S\ \textbf{end}\ T\rangle}{0} \;\Big\|\; \frac{T \parallel \langle S\ \langle\rangle\rangle}{0}$$

$$\frac{\langle \{\ x\ x_1 \ldots x_n\ \}\ T\rangle}{x = \xi \parallel \xi : \textbf{proc}\ \{\ y_1 \ldots y_n\ \}\ S\ \textbf{end}} \;\Big\|\; \frac{\langle S\{{}^{x_1}/_{y_1}\} \ldots \{{}^{x_n}/_{y_n}\}\ T\rangle}{x = \xi \parallel \xi : \textbf{proc}\ \{\ y_1 \ldots y_n\ \}\ S\ \textbf{end}}$$

$$\frac{\langle \{\ \texttt{Send}\ x\ y\ \}\ T\rangle}{x = \xi \parallel \xi : Q} \;\Big\|\; \frac{T}{x = \xi \parallel \xi : y; Q}$$

$$\frac{\langle \textbf{let}\ x = \{\ \texttt{Receive}\ y\ \}\ \textbf{in}\ S\ \textbf{end}\ T\rangle}{y = \xi \parallel \xi : Q; z \parallel z = w} \;\Big\|\; \frac{\langle S\{{}^{x'}/_{x}\}\ T\rangle}{y = \xi \parallel \xi : Q \parallel z = w \parallel x' = w} \quad \text{if } x' \text{ fr}$$

μOz reversible abstract machine

$$\frac{t[H]\langle \textbf{skip}\ C\rangle}{0} \parallel \frac{t[H\ \textbf{skip}]C}{0}$$

$$\frac{t[H]\langle \textbf{let}\ x = v\ \textbf{in}\ S\ \textbf{end}\ C\rangle}{0} \parallel \frac{t[H\ \ *x']\langle S\{^{x'}/_{x}\}\ \langle \textbf{esc}\ C\rangle\rangle}{x' = v}\ \text{if } x' \text{ fresh}$$

$$\frac{t[H]\langle \textbf{let}\ x = c\ \textbf{in}\ S\ \textbf{end}\ C\rangle}{0} \parallel \frac{t[H\ \ *x']\langle S\{^{x'}/_{x}\}\ \langle \textbf{esc}\ C\rangle\rangle}{x' = \xi \parallel \xi : c}\ \text{if } x', \xi \text{ fresh}$$

$$\frac{t[H]\langle \textbf{if}\ x\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{end}\ C\rangle}{x = \textbf{true}} \parallel \frac{t[H\ \textbf{if}(x)S_2]\langle S_1\ \langle \textbf{esc}\ C\rangle\rangle}{x = \textbf{true}}$$

$$\frac{t[H]\langle \textbf{if}\ x\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{end}\ C\rangle}{x = \textbf{false}} \parallel \frac{t[H\ \textbf{if}(x)S_1]\langle S_2\ \langle \textbf{esc}\ C\rangle\rangle}{x = \textbf{false}}$$

μOz reversible abstract machine

$$\frac{t[H]\langle \textbf{let } x = \texttt{NewPort} \textbf{ in } S \textbf{ end } C\rangle}{0} \;\Bigg\|\; \frac{t[H \;\; * x']\langle S\{^{x'}/_{x}\}\ \langle \textbf{esc } C\rangle\rangle}{x' = \xi \parallel \xi : \bot|\bot} \ \text{if } x', \xi \text{ fresh}$$

$$\frac{t[H]\langle \textbf{thread } S \textbf{ end } C\rangle}{0} \;\Bigg\|\; \frac{t[H \;\; * t']C \parallel t'[\bot]\langle S\ \langle\rangle\rangle}{0} \ \text{if } t' \text{ fresh}$$

$$\frac{t[H]\langle\{\ x\ (x_i)_1^n\ \}\ C\rangle}{x = \xi \parallel \xi : \textbf{proc } \{\ (y_i)_1^n\ \}\ S \textbf{ end}} \;\Bigg\|\; \frac{t[H\ \{\ x\ (x_i)_1^n\ \}]\langle S(\{^{x_i}/_{y_i}\})_1^n\ \langle \textbf{esc } C\rangle\rangle}{x = \xi \parallel \xi : \textbf{proc } \{\ (y_i)_1^n\ \}\ S \textbf{ end}}$$

$$\frac{t[H]\langle\{\ \texttt{Send}\ x\ y\ \}\ C\rangle}{x = \xi \parallel \xi : K|K_h} \;\Bigg\|\; \frac{t[H \;\; \uparrow x]C}{x = \xi \parallel \xi : t\!:\!y; K|K_h}$$

$$\frac{t[H]\langle \textbf{let } y = \{\ \texttt{Receive}\ x\ \} \textbf{ in } S \textbf{ end } C\rangle}{\theta \parallel \xi : K; t'\!:\!z|K_h} \;\Bigg\|\; \frac{t[H \;\; \downarrow x(y')]\langle S\{^{y'}/_{y}\}\ \langle \textbf{esc } C\rangle\rangle}{\theta \parallel \xi : K|t'\!:\!z, t; K_h \parallel y' = w}$$

$$\frac{t[H]\langle \textbf{esc } C\rangle}{0} \;\Bigg\|\; \frac{t[H \ \textbf{esc}]C}{0}$$

μOz reversible abstract machine

$$\frac{\langle \textbf{thread } S \textbf{ end } T\rangle}{0} \;\Bigg\|\; \frac{T \parallel \langle S\ \langle\rangle\rangle}{0}$$

$$\frac{t[H]\langle \textbf{thread } S \textbf{ end } C\rangle}{0} \;\Bigg\|\; \frac{t[H\ * t']C \parallel t'[\perp]\langle S\ \langle\rangle\rangle}{0} \text{ if } t' \text{ fresh}$$

μOz reversible abstract machine

$$\frac{\langle \textbf{let } x = \texttt{NewPort} \textbf{ in } S \textbf{ end } T\rangle}{0} \;\Bigg\|\; \frac{\langle S\{^{x'}/_{x}\}\ T\rangle}{x' = \xi \parallel \xi : \bot} \text{ if } x', \xi \text{ fresh}$$

$$\frac{t[H]\langle \textbf{let } x = \texttt{NewPort} \textbf{ in } S \textbf{ end } C\rangle}{0} \;\Bigg\|\; \frac{t[H\ * x']\langle S\{^{x'}/_{x}\}\ \langle \textbf{esc } C\rangle\rangle}{x' = \xi \parallel \xi : \bot|\bot} \text{ if } x', \xi \text{ fresh}$$

Results

Theorem (Linear overhead for reversibility)

Assume $(t[\perp]\langle S\ \langle\rangle\rangle, 0) \rightarrow^{n}_{vm} (M, \theta)$, *where* $\rightarrow^{n}_{vm}$ *denotes* n $\rightarrow_{vm}$ *steps. Then*

$$\textbf{overhead}(M, \theta) \leq n \cdot \textbf{stsize}(S)$$

Theorem (Linear space lower bound for reversibility)

The amount of information to be stored to ensure causally consistent reversibility of μOz programs is at least linear in the number of execution steps.

Linear lower bound

Example

```
let a = NewPort in
let x = true in
let y = false in
let p1 = proc{} {Send a x} {p1} end in
let p2 = proc{} {Send a y} {p2} end in
let p3 = proc{} let z = {Receive a} in {p3} end in
  thread {p1} end
  thread {p2} end
  thread {p3} end
```

Taking stock

- Inherent linear overhead to reversibility in a concurrent setting
    - Tied to non-determinism
- Scope for exploiting determinism
    - towards classical tradeoff: granularity of rollback / space overhead

Roadmap

Much work remains

- Behavioral theory for croll-π
  - contextual equivalence, bisimulation proof techniques, etc
- Primitives for reversibility control and compensation
  - e.g. encoding BPM compensation schemes in variants of croll-π
- Primitives for general transactional constructs
  - what about isolation ?
- Taking localities into account
  - modularity constructs (components), distribution
- Reversible language design and implementation
  - exception handling as rollback/compensation, contracts for reversible components, etc
  - garbage collection and debugging in a reversible language
  - multicore reversible virtual machine, associated algorithms, etc

Thank you for your attention

Questions ?