Croll-$\pi$, a reversible calculus with compensations

Ivan Lanese
Computer Science Department
Focus research group
University of Bologna/INRIA
Bologna, Italy

Joint work with Michael Lienhardt (PPS), Claudio Antares Mezzina (Trento), Jean-Bernard Stefani (INRIA) and Alan Schmitt (INRIA)
Roadmap

- The approach
- Croll-$\pi$
- Applications
- Conclusions
Roadmap

- The approach
- Croll-π
- Applications
- Conclusions
Uncontrolled perfect reversibility

- RCCS, CCSk, $\rho\pi$, reversible structures all feature uncontrolled perfect reversibility
  - Uncontrolled since the process can go nondeterministically backward or forward
  - Perfect since a backward step perfectly undoes a forward step

- Some approaches feature some form of control of reversibility
  - Irreversible actions in RCCS
  - Roll-$\pi$ roll operator
  - Controller process in CCSk

- Reversibility is still perfect in all these approaches
Drawbacks of perfect reversibility

- With perfect reversibility one goes back to a past state
- From this state the same computation can restart
  - Probably leading to the same rollback
  - Again and again
- The programmer has no tool to avoid this
- We want to provide such a tool
  - Using a compensation mechanism
- We build on top of roll-π
Do you remember roll-π?

- HOπ extended with an explicit roll operator
- The roll has a parameter γ that refers to a past trigger
- The roll undoes all the actions depending on the communication done by the trigger
- Idea: in case of error I undo the computation that caused the error
Compensations

- From database theory and business processes
  - Piece of code to recover from an error

- For us, a piece of code allowing to move
  - From the past state reached by the rollback
  - To a slightly different state
  - Keeping into account the past try
  - Trying to avoid to repeat the same error

- We call croll-\(\pi\) the language integrating compensations into roll-\(\pi\)
Messages with compensations

- Instead of roll-π messages $a<P>$ we use messages with compensations
  - $a<P>%0 : try a<P> then stop trying$
  - $a<P>%b<Q>%0 : try a<P> then b<Q> then stop trying$
- If the message with compensation is the target of a rollback, it is replaced by its compensation
- Minimal change to roll-π syntax and semantics
- The expressive power increases considerably
Roadmap

- The approach
- Croll-π
- Applications
- Conclusions
Croll-π syntax

\[
P, Q ::= 0 \mid X \mid \nu a. P \mid (P \mid Q) \mid a(X) \triangleright_\gamma P \mid a(P)\!\div C \mid \text{roll } k \mid \text{roll } \gamma
\]
\[
M, N ::= 0 \mid \nu u. M \mid (M \mid N) \mid k : P \mid [\mu; k] \mid k \prec (k_1, k_2)
\]
\[
C ::= a(P) \div 0 \mid 0
\]
\[
\mu ::= (k_1 : a(P) \div C) \mid (k_2 : a(X) \triangleright_\gamma Q)
\]

- \text{HOπ processes + roll + } \gamma + \text{ compensations}
- Configurations featuring memories and causal dependencies
Croll-π semantics

\[ \mu = (k_1:a \langle P \rangle \triangleright C) | (k_2:a(X) \triangleright_\gamma Q_2) \]

\[ (k_1:a \langle P \rangle \triangleright C') | (k_2:a(X) \triangleright_\gamma Q_2) \rightarrow \nu k. (k:Q_2 \{ P, k / x, \gamma \}) || [\mu; k] \]

\[ k <: N \text{ complete}(N || [\mu; k] | (k_r: \text{roll } k)) \]

\[ \mu' = \text{xtr}(\mu) \]

\[ N || [\mu; k] | (k_r: \text{roll } k) \rightarrow \mu' | N \not\vdash k \]

\[ k:P | Q \rightarrow \nu k_1, k_2. k < (k_1, k_2) | k_1:P | k_2:Q \]

\[ k: \nu a. P \rightarrow \nu a. k:P \]

- Instead of restoring the initial configuration we replace the message with its compensation
- \text{xtr}(a\langle P\rangle \%C)=C
Croll-$\pi$ example

\[ k_1 : a(0) \bowtie b(0) \quad k_2 : a(X) \triangleright_\gamma b(roll \ \gamma) \quad k_3 : b(X) \triangleright c(0) \mid X \]
Croll-π example

\[ k_1 : a(0) \% b(0) \quad k_2 : a(X) \triangleright_\gamma b(\text{roll } \gamma) \quad k_3 : b(X) \triangleright c(0) \mid X \]

\[ [k_1 : M \mid k_2 : N ; k] \]

\[ k : b(\text{roll } k) \]
Croll-π example

\[ k_1 : a(0) \% b(0) \quad k_2 : a(X) \triangleright_\gamma b(\text{roll } \gamma) \quad k_3 : b(X) \triangleright c(0) \mid X \]

\[ [k_1 : M \mid k_2 : N; k] \]

\[ k : b(\text{roll } k) \]

\[ [k : M_1 \mid k_3 : N_1 ; k_4] \]

\[ k_5 : c(0) \quad k_6 : \text{roll } k \]
Croll-π example

\[ k_1 : a(0) \% b(0) \quad k_2 : a(X) \triangleright_{\gamma} b(\text{roll } \gamma) \quad k_3 : b(X) \triangleright c(0) \mid X \]

\[ [k_1 : M \mid k_2 : N; k] \]

\[ k : b(\text{roll } k) \]

\[ [k : M_1 \mid k_3 : N_1; k_4] \]

\[ k_5 : c(0) \]

\[ k_6 : \text{roll } k \]
Croll-$\pi$ example

\[ k_1 : b\langle 0 \rangle \quad k_2 : a(X) \triangleright_\gamma b\langle \text{roll } \gamma \rangle \quad k_3 : b(X) \triangleright c\langle 0 \rangle \mid X \]
Roadmap

- The approach
- Croll-\(\pi\)
- Applications
- Conclusions
And now?

- We have to test the expressive power of croll-$\pi$
- Can we programme interesting applications exploiting rollback and compensations?
- Compensations allow to specify what to do after a rollback
- Croll-$\pi$ is strictly more expressive than roll-$\pi$
Messages with compensations are robust

- We can encode different idioms:
  - General compensations: not only messages
  - Finite retry: try n times
  - Endless retry: try forever (same as roll-π)
  - Triggers with compensations: we can attach compensations to triggers instead of to messages
Triggers with compensations

\[ a(X) \triangleright_{\gamma} Q \div C \]

- Compensation replaces trigger instead of message
  \[ (\{a(X) \triangleright_{\gamma} Q \div C\}_{tc} = \nu c, d. \bar{c} \div \bar{d} \div 0 \mid (c \triangleright_{\gamma} a(X) \triangleright \{Q\}_{tc}) \mid (d \triangleright \{C\}_{tc}) \]

- Messages with compensations and triggers with compensations can coexist

- Makes the framework more symmetric
What can we model?

- **Interesting problems**
  - State space exploration with backtracking: 8 queens problem
  - Error handling scenario: Automotive case study from Sensoria project

- **Can we recover/improve existing techniques?**
  - Interacting transactions from Hennessy et al. [CONCUR 2010]
  - Software transactional memories
    » Different approaches
    » We started from the process calculus account by Acciai et al. [ESOP’07]
Eight queens problem

- Queens are interacting processes
- Queen $i$ tries the 8 positions on column $i$ then fails by notifying the previous queen
- If it succeeds it notifies the next queen, and waits for failures from her

$$Q_i \overset{\Delta}{=} c_1(x_1) \Join \ldots \Join c_{i-1}(x_{i-1}) \Join p_i(i, 1) \div \ldots \div p_i(i, 8) \div f_i(0) \div 0$$

\[ \text{if} \ err(x_1, x) \ \text{then roll } \gamma_i \ \text{else} \ldots \ \text{if} \ err(x_{i-1}, x) \ \text{then roll } \gamma_i \]

\[ \text{else} \ !c_i(x) \div 0 \ | \ f_{i+1}(y) \Join \text{roll } \gamma_i \]

$$err((x_1, x_2), (y_1, y_2)) \overset{\Delta}{=} (x_1 = y_1 \lor x_2 = y_2 \lor |x_1 - y_1| = |x_2 - y_2|)$$

- We also have a more concurrent solution
Interacting transactions

- Allow a transaction to interact with the environment
- In case of rollback the effects on the environment are undone

\[
\overline{a} | a. P \rightarrow P \quad \text{\( k \notin \text{fn}(R) \)} \quad [P \triangleright_k Q] \mid R \rightarrow [P | R \triangleright_k Q | R]
\]

\[
[P | \text{co } k \triangleright_k Q] \rightarrow P \quad [P \triangleright_k Q] \rightarrow Q
\]

- Parts of the environment moved inside the transaction to interact with it
  - Also saved in the compensation
Modeling interacting transactions

- A transaction is a computation originated by a trigger labeled by \( \gamma \)
- Roll \( \gamma \) used as an abort
- Commit by disabling roll \( \gamma \)

\[
([[[P \triangleright_l Q]]]_t) = v a, c. \overline{a} \div c \div 0 \mid a \triangleright_{\gamma} (v l. ([P]_t) \mid l(\text{roll } \gamma) \mid l(X) \triangleright X) \mid c \triangleright ([Q]_t)
\]

- More precise than Hennessy
- Only parts of the environment that interacted with the transaction are actually rollbacked
Roadmap

- The approach
- Croll-π
- Applications
- Conclusions
Summary

- A framework supporting controlled rollback and compensations
- Some interesting applications
  - 8 queens problem
  - Interacting transactions
  - Software transactional memories
Future work

- Is this enough for programming recoverable systems?
  - Some good hints, but more work is needed

- A known limitation
  - We have no information on what caused the roll
  - The compensation does not depend on the reason of the failure
  - A mechanism to keep this information would be helpful

- Studying the behavioral theory
  - To reason about the encodings
  - To have an axiomatic theory
  - History information makes things more complex
Finally

Thanks!

Questions?