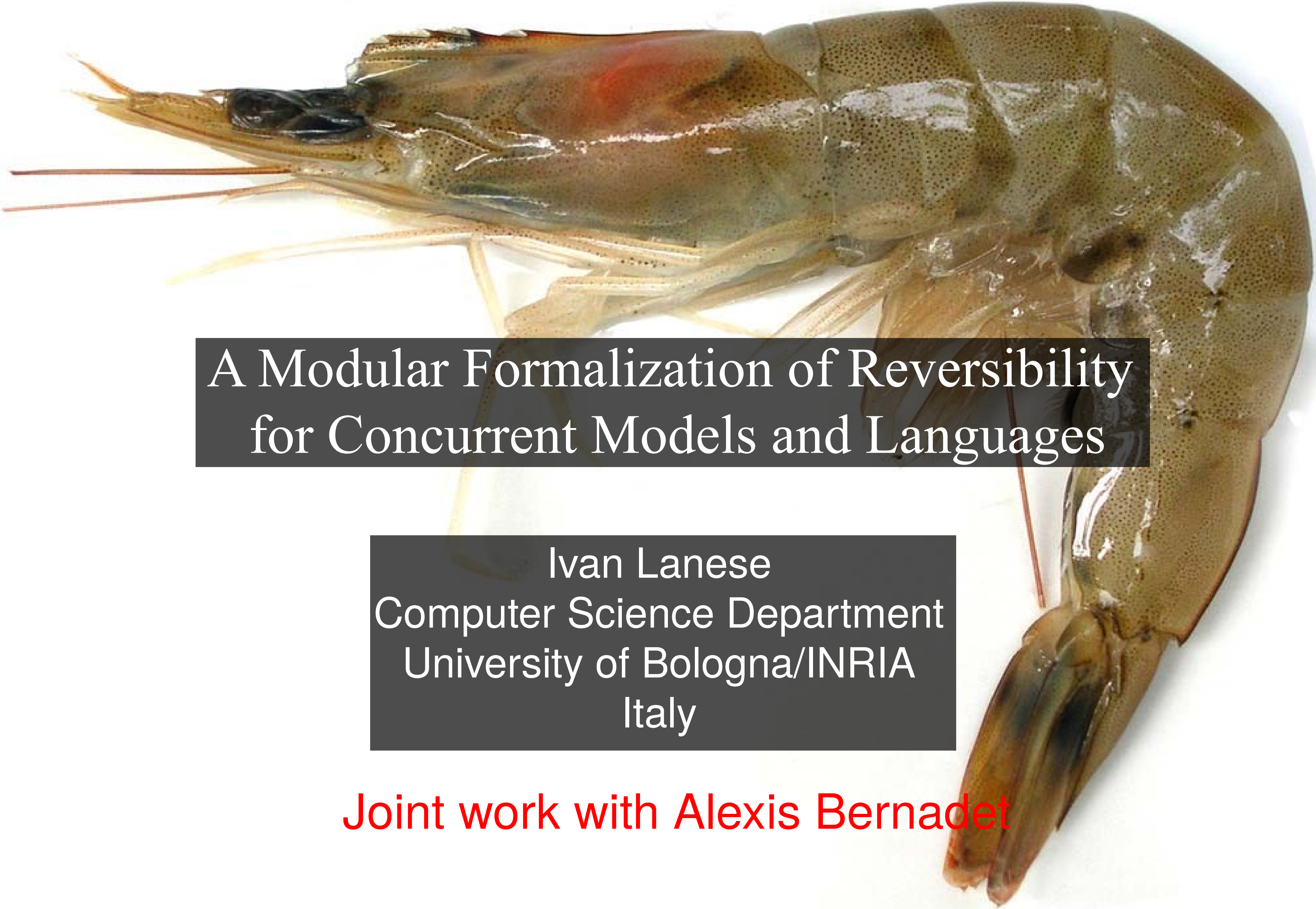


A Modular Formalization of Reversibility for Concurrent Models and Languages

Ivan Lanese
Computer Science Department
University of Bologna/INRIA
Italy

Joint work with Alexis Bernadet

Map of the talk

- Reversibility
- A simple case
- Label refinement
- Conclusion

Map of the talk

- Reversibility
- A simple case
- Label refinement
- Conclusion

What is reversibility?

The possibility of executing a computation both in the standard, forward direction, and in the backward direction, going back to past states

- Reversibility everywhere
  - chemistry/biology
  - quantum computing
  - state space exploration
  - debugging
  - optimistic simulation
  - low-power computing
  - …

Reverse execution of a sequential program

- Recursively undo the last action
  - Computations are undone in reverse order
  - To reverse A;B first reverse B, then reverse A

- We want the Loop Lemma to hold
  - From state S, doing A and then undoing A should lead back to S
  - From state S, undoing A (if A is in the past) and then redoing A should lead back to S

Avoiding loss of information

- Undoing computational actions may not be easy
  - Computational actions may cause loss of information
  - X = 5 causes the loss of the past value of X
- Restrict to languages that never lose information
  - X = X + 1 does not lose information
- Take languages that would lose information, and save this information
  - X = 5 becomes reversible by recording the old value of X

Reversibility and concurrency

- The sequential definition, recursively undo the last action, is no more applicable

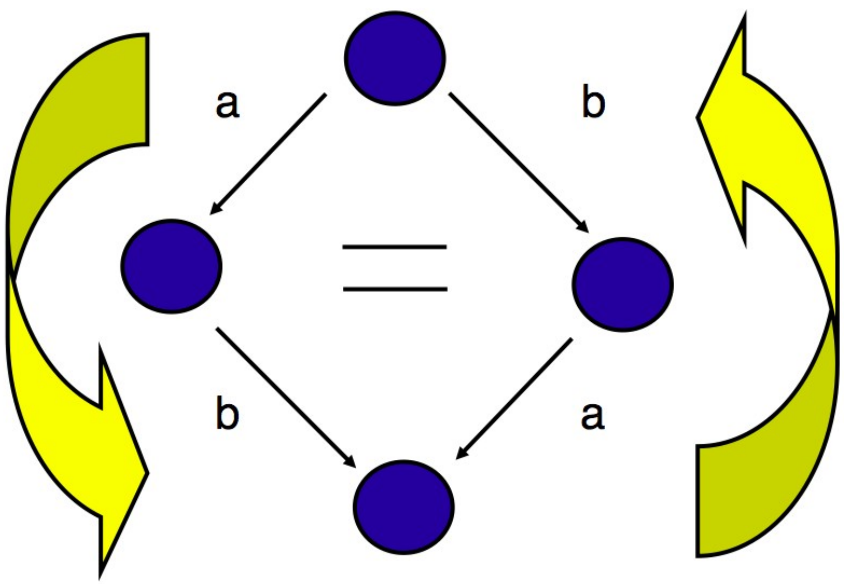
- Which is the last action in a concurrent setting?
  - Executions of many actions may overlap
  - For sure, if an action A caused an action B, A could not be the last one

- Causal-consistent reversibility: recursively undo any action whose consequences (if any) have already been undone

Causal-consistent reversibility in practice

- Two sequential actions should be undone in reverse order

- Two concurrent actions can be undone in any order
  - Choosing an interleaving for them is an arbitrary choice
  - It should have no impact on the possible reverse behaviors



Our aim

- Many causal-consistent reversible extensions of specific concurrent languages (CCS, π-calculus, klaim, ...) exist in the literature
- Very little on how to define the extension, given a language
- We look for a general and modular way of doing this
- We prefer simplicity and correctness by construction over memory or time efficiency

Map of the talk

- Reversibility
- A simple case
- Label refinement
- Conclusion

A simple case

- Take a system represented by an LTS with transitions $M \xrightarrow{\alpha} M'$
- Assume the LTS is
  - Deterministic: given M and α there is a unique M'
  - Codeterministic: given M' and α there is a unique M
- Each computation is fully described by its final state M_n and its sequence of labels $\alpha_1, \ldots, \alpha_n$

A simple case: reversible LTS

- Configurations: (L,M) where L is a sequence of labels such that there exists M' with $M' \xrightarrow{L} M$

- Forward transitions: $(L,M) \xrightarrow{\alpha} ((L,\alpha),M')$ if $M \xrightarrow{\alpha} M'$

- Backward transitions: $((L,\alpha),M') \xrightarrow{\alpha^{-1}} (L,M)$ if $M \xrightarrow{\alpha} M'$

Properties

- The Loop Lemma holds:
  $(L,M) \xrightarrow{\alpha} (L',M')$ iff $(L',M') \xrightarrow{\alpha^{-1}} (L,M)$
- The reversible LTS is a conservative extension of the given one
- This is not causal-consistent reversibility, just sequential reversibility
  - Only the last action can be undone
- In many relevant cases the starting LTS is not deterministic and/or not codeterministic

Adding concurrency

- We need to know which actions are concurrent
  - The same formalism may have different concurrency models
- We require a symmetric independence $\perp$ relation on labels
- Independent actions should satisfy the co-diamond property

  If $M_1 \xrightarrow{\alpha} M_2$, $M_2 \xrightarrow{\beta} M_3$ and $\alpha \perp \beta$ then there exists M'_2 such that $M_1 \xrightarrow{\beta} M'_2$ and $M'_2 \xrightarrow{\alpha} M_3$
- Standard labels may not contain enough information to define $\perp$ in order to capture the desired concurrency model
  - In CCS $a.b+b.a$ and $a|b$ have the same labels

A causal-consistent reversible LTS

- Configurations: ([L],M) where
  - L is a sequence of labels such that there exists M' with $M' \xrightarrow{L} M$
  - [L] is the equivalence class of L w.r.t. a relation allowing one to swap independent actions
- Forward transitions: $([L],M) \xrightarrow{\alpha} ([L,u],M')$ if $M \xrightarrow{\alpha} M'$
- Backward transitions: $([L,\alpha],M') \xrightarrow{\alpha^{-1}} ([L],M)$ if $M \xrightarrow{\alpha} M'$

Properties

- The Loop Lemma holds
- The Causal Consistency theorem holds
  - It characterizes causal-consistent reversibility
- The reversible LTS is a conservative extension of the given one
- In many relevant cases the starting LTS is not deterministic and/or not codeterministic
- We may not have enough information in the labels to define the independence relation as desired

Causal Consistency theorem

- Causal equivalence is an equivalence relation on computations allowing one to:
  - Swap independent actions
  - Simplify inverse actions
- Causal Consistency theorem: two computations are coinitial and cofinal iff they are causal equivalent
  - Causal equivalent computations lead to the same state
    - Have the same backward behavior
  - Computations which are not causal equivalent produce different history/causal information
    - Have different backward behavior

Map of the talk

- Reversibility
- A simple case
- Label refinement
- Conclusion

Label refinement

- The LTS may be not deterministic or not codeterministic
- We may not have enough information to define the desired independence relation
- We solve both the problems at once:

  we refine the labels so to have enough information
- Adding too much information to the labels may forbid some concurrency model
  - It may not be possible to preserve the labels when swapping independent actions (co-diamond property)

Label refinement: definition

- We want
  - a new set of labels u
  - an LTS with the same terms M but labels u which is deterministic and codeterministic
  - an interpretation [[u]]=α such that
    $M \xrightarrow{u} M'$ iff $M \xrightarrow{\alpha} M'$ (correctness)
- We can now apply the construction to our reversible LTS

Label refinement: example

- Refinement is always possible

- For each $M \xrightarrow{\alpha} M'$ take $u = (M,\alpha,M')$

- … but not always useful: transitions can (almost) never commute preserving labels

Label refinement in general

- No automatic way of finding a good refinement
- We should add enough information to be deterministic, codeterministic, and distinguish independent actions
- We should not add too much information to allow independent actions to commute without changing labels
- We will see action refinement for CCS
- The paper also discusses action refinement for concurrent X-machines

The case of CCS

- Standard CCS semantics
  - is not deterministic
  - is not codeterministic
  - does not provide enough information to define independence
- We consider a label refinement close to the causal semantics of Boudol and Castellani

Label refinement for CCS

- We define the following refined labels:
  $u,v ::= ((\alpha_j,P_j)_{j\in I},i) \mid (u|\bullet) \mid (\bullet|u) \mid (u|v) \mid \upsilon a.u \mid recX.P$
- Action interpretation is as follows:

$$[[((\alpha_j,P_j)_{j\in I},i)]] = \alpha_i \qquad [[recX.P]] = \tau$$

$$[[(u|\bullet)]] = [[u]] \qquad [[(u|\bullet)]] = [[u]]$$

$$[[(u|v)]] = \tau \qquad [[\upsilon a.u]] = [[u]]$$

- Labels are independent if originated from different threads
  - E.g., $(u|\bullet) \perp (\bullet|v)$

An example

- $a.b.0|\overline{b}.c.0 \xrightarrow{(a,b.0),1|\bullet} b.0|\overline{b}.c.0 \xrightarrow{\bullet|(\overline{b},c.0),1} b.0|c.0$
  The two actions can be reversed in any order since $((a,b.0),1|\bullet)$, $(\bullet|(\overline{b},c.0),1)$ and $(\bullet|(\overline{b},c.0),1,)$ $(a,b.0),1|\bullet)$ are equivalent
- $a.b.0|\overline{b}.c.0 \xrightarrow{(a,b.0),1|\bullet} b.0|\overline{b}.c.0 \xrightarrow{(b,0),1|(\overline{b},c.0),1} 0|c.0$
  The two actions need to be reversed in any order since they are causally dependent

The final recipe

- Define a suitable refinement of the given LTS
- Choose an independence relation
- Apply the construction to get a causal-consistent reversible LTS extending the given one
- Get for free many relevant properties
  - Loop Lemma and Causal-Consistency theorem

Map of the talk

- Reversibility
- A simple case
- Label refinement
- Conclusion

Summary

- We presented a modular way to define a causal-consistent semantics of a given formalism
  - It ensures by construction the Loop Lemma and the Causal Consistency theorem
- We instantiated it on two case studies
  - CCS
  - Concurrent X-machines

Future work

- Applying our framework to other formalisms
- Defining a causal-consistent reversible semantics for a mainstream language
- Find fully automatic ways for defining a causal-consistent reversible semantics given a forward one

Thanks!

Questions?