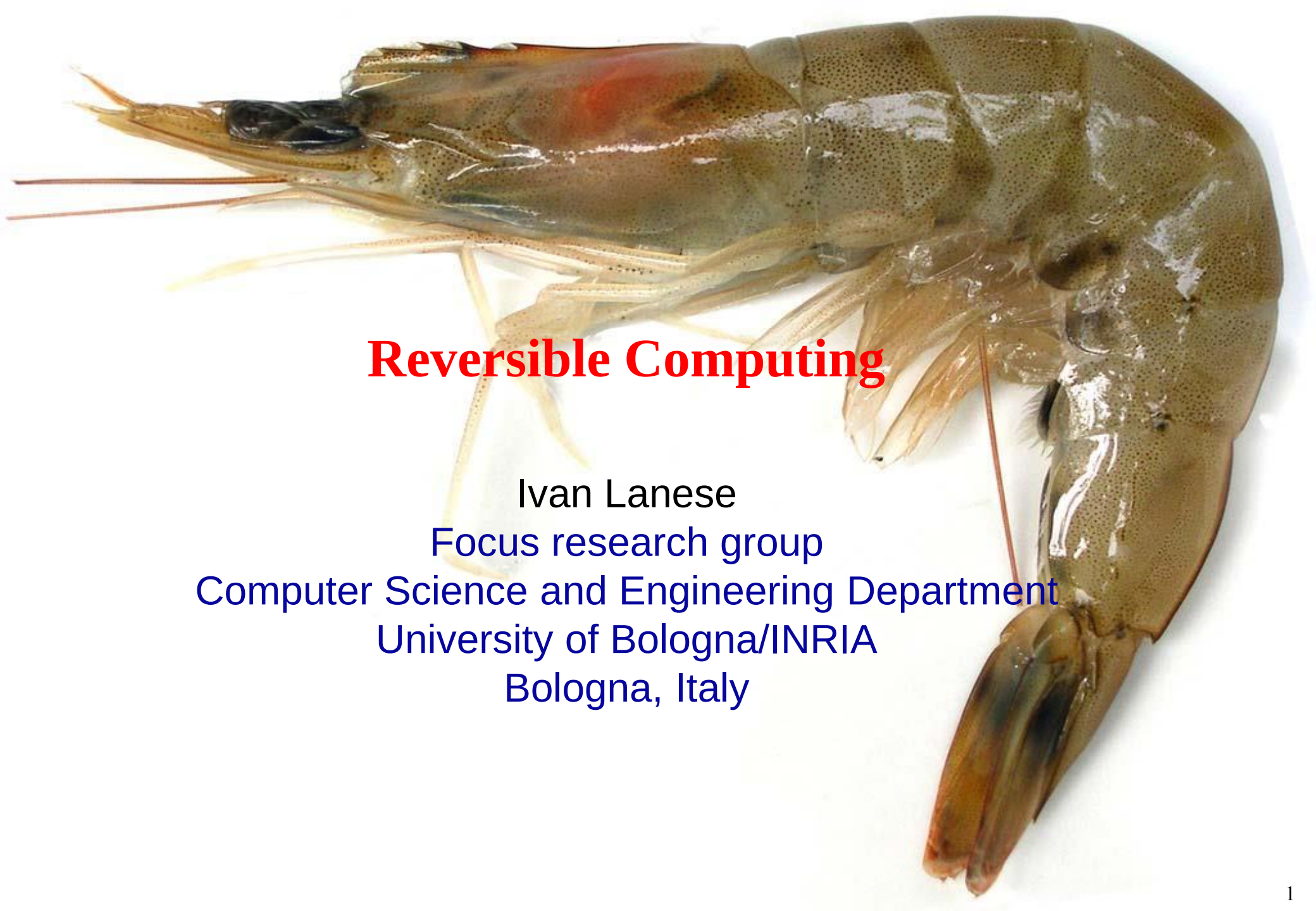


# Reversible Computing

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

# Contributors

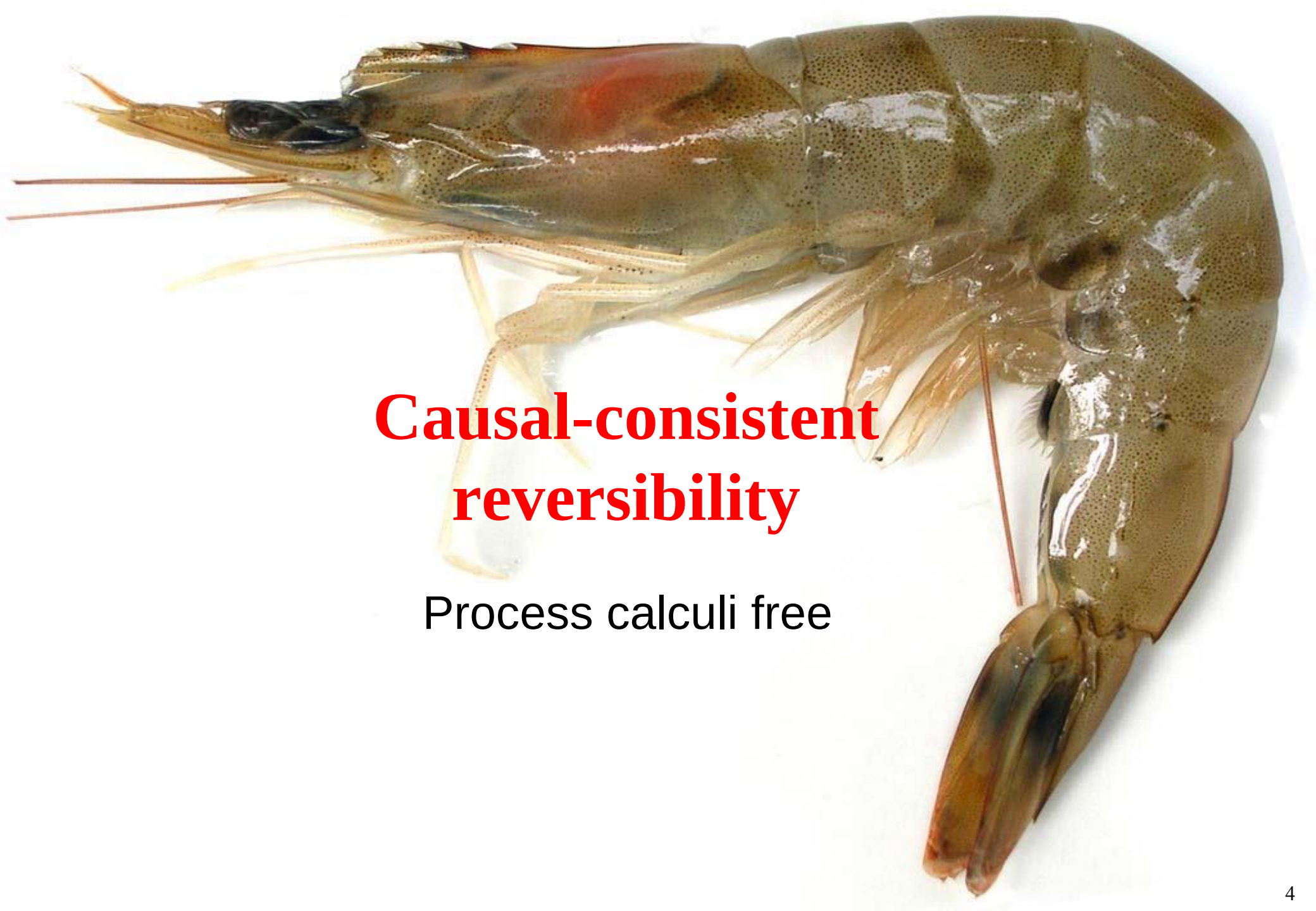
- Elena Giachino (Bologna), Michael Lienhardt (PPS), Claudio Antares Mezzina (FBK Trento), Jean-Bernard Stefani (INRIA Grenoble), Alan Schmitt (INRIA Rennes)

- Other groups working on similar topics
  - Jean Krivine (PPS), Daniele Varacca (PPS)
  - Irek Ulidowski (Leicester), Iain Phillips (Imperial)

- The French ANR Project REVER
  - INRIA Grenoble, PPS, CEA, Bologna

# Plan of the course

1. Causal-consistent reversibility
2. Defining uncontrolled causal-consistent reversibility
3. Controlling reversibility
4. Specifying alternatives
5. Transactions
6. Causal-consistent debugging



# Causal-consistent reversibility

Process calculi free

# What is reversibility?

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

- What does it mean to go backward?
- If from state $S_1$ I go forward to state $S_2$, then from state $S_2$ I should be able to go back to state $S_1$

# Reversibility everywhere

- Reversibility widespread in the world
  - Chemistry/biology
  - Quantum phenomena
  - ...
- Reversibility for modelling
- Reversibility for programming
  - DNA circuits
  - Quantum computers
  - State space exploration
  - View-update problem

# Reversibility in chemistry/biology

- Most of the chemical and/or biological phenomenons are reversible
- Direction of execution depends on environmental conditions such as temperature or pressure
- RCCS, the first reversible process calculus, was devised to model biological systems
  [Vincent Danos, Jean Krivine: Reversible Communicating Systems. CONCUR 2004]
- A reversible language for programming biological systems:
  [Luca Cardelli, Cosimo Laneve: Reversible structures. CMSB 2011]

# Reversibility in quantum computing

- Most quantum operations are reversible
  - Essentially the ones that do not involve observation
- Quantum programming languages feature many reversible operators
- Survey of languages for quantum computing [Simon J. Gay: Quantum programming languages: survey and bibliography. Mathematical Structures in Computer Science (2006)]

# State space exploration

- While exploring a state space towards a solution one may find a dead end
- Need to backtrack to find a solution
- This is the standard mechanism in Prolog
- State space exploration much easy in a reversible language
  - No need to program backtracking

# View-update problem

- Views allow one to access (part of) a data structure
  - Views of databases
- The user may want to modify the view
- How to reflect the changes on the data structure?
- Easy if the view is generated by a reversible language
  - Lenses
- A survey of the approach is in
  [Benjamin C. Pierce et al.: Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. ACM Trans. Program. Lang. Syst. 29(3) (2007)]

# Our aim

- We want to exploit reversibility for programming reliable concurrent/distributed systems
- Understanding reversibility is the key to
  - Understand existing patterns for programming reliable systems
  - Combine and improve them
  - Develop new patterns

# The idea of the approach

- To make a system reliable we want to avoid “bad” states
- If a bad state is reached, reversibility allows to go back to some past state
- Far enough, so that the decisions leading to the bad state have not been taken yet
- When I restart computing forward, I should try new directions

# Reversibility for reliable systems

- Reversibility seems related to some patterns for programming reliable systems
- Checkpointing
  - I save the state of the program to restore it in case of errors
- Rollback-recovery
  - I combine checkpointing with logs to recover a program state
- Transactions
  - Computations which are executed all or nothing
  - In case of error their effect should be undone
  - Both in database systems (ACID transactions) and in service oriented computing (long running transactions)

# Some hidden reversibility?

- Application undo
  - Allows us to undo a wrong command in our favorite editor
- Backup
  - Allows to go back to a past version of a file
- SVN and the like
  - More refined techniques to go back to past versions of files
- Understanding reversibility will shed some light on these mechanisms too?

# The current state of the art

- A bag of tricks to face different problems
- No clue on whether and how the different tricks compose
- No unifying theory for them
- Many approaches have poor theoretical foundations

- We want to face this challenge

# Reverse execution of a sequential program

- Recursively undo the last step
  - Computations are undone in reverse order
  - To reverse A;B reverse B, then reverse A
- First we need to undo single computation steps
- 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

# Undoing computational steps

- Not necessarily easy
- Computation steps may cause loss of information
- X=5 causes the loss of the past value of X
- X=X+Y causes no loss of information
  - Old value of X can be retrieved by doing X=X-Y
- X=X*Y causes the loss of the value of X only if Y is 0

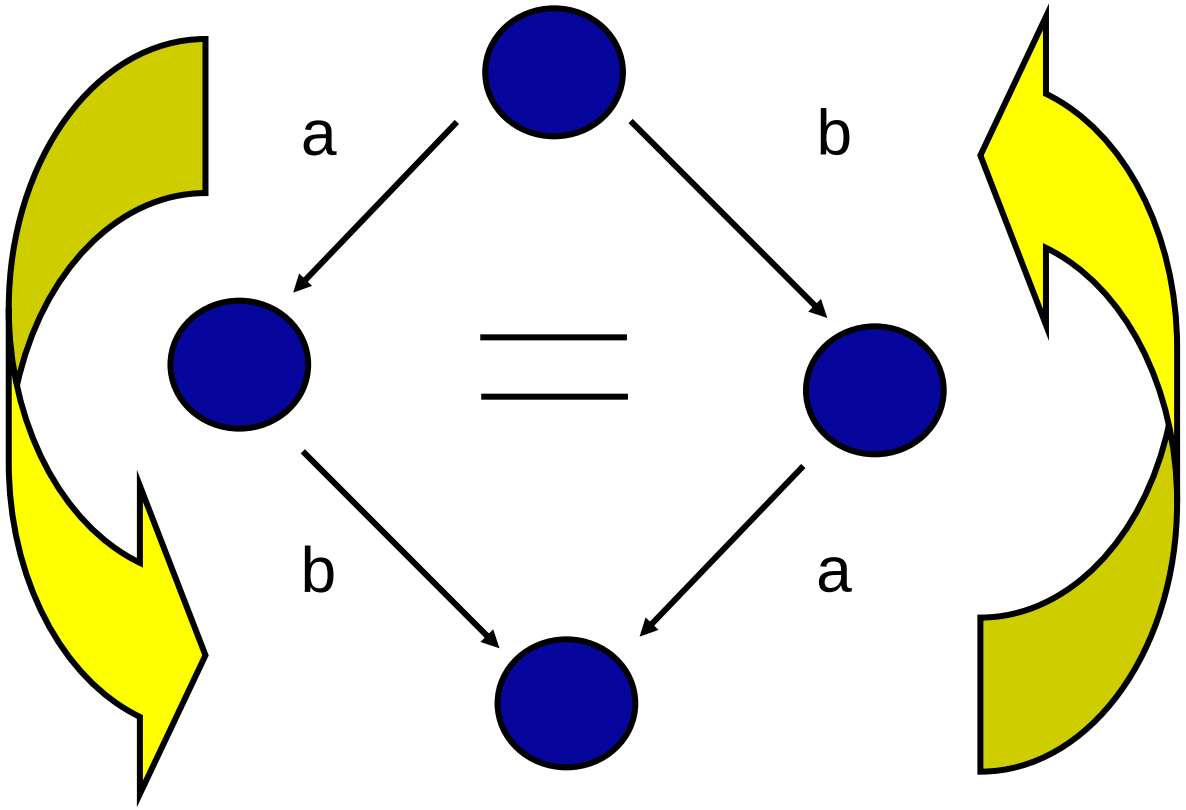
# Different approaches to reversibility

- Undoing steps one by one
  - Considering languages which are reversible
    - » Featuring only actions that cause no loss of information
  - Taking a language which is not reversible and make it reversible
    - » One should save information on the past configurations
    - » X=5 becomes reversible by recording the old value of X
- Saving a past state and redoing the same computation from there

# Reversibility and concurrency

- In a sequential setting, recursively undo the last step
- Which is the last step in a concurrent setting?
- Many possibilities
- 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



# Non-determinism versus concurrency

- Causal-consistent reversibility allows one to distinguish non-determinism from concurrency
- Two sequential actions whose order is chosen nondeterminstically should be reversed in reverse order
- Two concurrent actions can be reversed in any order
  - Choosing an interleaving for them is an arbitrary choice
  - It should have no impact on the possible reverse behaviors

# History information

- To reverse actions we need to store some history information
- Different threads are reversed independently
- It makes sense to attach history information to threads
- History information should trace where we come from
  - X=5 destroys the old value of X
  - We need to store the old value of X to know which state we come from

# Causal history information

- We need to remember causality information
- If thread $T_1$ sent a message to thread $T_2$ then $T_1$ cannot reverse the send before $T_2$ reverses the receive
  - We need to remember the receiver of all the messages we sent
  - We need to remember the sender of all the messages we received

# Causal equivalence

- According to causal-consistent reversibility
  - Changing the order of execution of concurrent actions should not make a difference
  - Doing an action and then undoing it should not make the difference (Loop Lemma)
- Two computations are called causal equivalent if they are equal up to the transformations above

# Causal consistency theorem

- Causal equivalent computations should
  - Lead to the same state
  - Produce the same history information
- Computations which are not causal equivalent
  - Should not lead to the same state
  - Otherwise we would erroneously reverse at least one of them in the wrong way
  - If in a non reversible setting they would lead to the same state, we should add history information to distinguish the states

# Example

- If x>5 then
  y=6;x=2
  else
  x=2;y=6
  endif
- Two possible computations
- The two possible computations lead to the same state
- From the causal consistency theorem I need history information to distinguish them
  - At least I should trace the chosen branch

# What we know

- We have some idea about how to define a causal-consistent reversible variant of a concurrent language
  - We need to satisfy the Loop Lemma
  - We need to satisfy the Causal Consistency Theorem
- More technicalities are needed to do it
  - We will leave it for tomorrow
- We continue to explore reversibility in an informal way

# What we don’t know

- We know only uncontrolled reversibility
- We have a language able to go both back and forward
- When to go backward and when to go forward?
- Just non-deterministic is not a good idea
  - The program may go back and forward between the same states forever
  - If a good state is reached, the program may go back and lose the computed result
- We need some form of control for reversibility

# Reversibility control


- One may imagine different ways of controlling reversibility
- We will show some possible alternatives
- We will try to categorize them
- We will try to understand the state space of the possible mechanisms
- The choice of the good mechanism depends on the intended application field

# A taxonomy for reversibility control

- Categorization according to who controls the reversibility
- Three different possibilities
  - Internal control: reversibility is controlled by the programmer
  - External control: reversibility is controlled by the environment
  - Semantic control: reversibility control is embedded in the semantics of the language

# Internal control

- Reversibility is controlled by the programmer
- Explicit operators to specify whether to go backward and whether to go forward
- We have different possibilities
  - Irreversible actions
    [Vincent Danos, Jean Krivine: Transactions in RCCS. CONCUR 2005]
  - **Roll** operator
    [Ivan Lanese, Claudio Antares Mezzina, Alan Schmitt, Jean-Bernard Stefani: Controlling Reversibility in Higher-Order Pi. CONCUR 2011]

# Irreversible actions

- Execution is non-deterministically backward or forward
- Some actions, once done, cannot be undone
  - This allows to make a computed result permanent
  - They are a form of commit
- Still most programs are divergent
- Suitable to model biological systems
  - Most reactions are reversible
  - Some are not

# **Roll** operator

- Normal execution is forward
- Backward computations are explicitly required using a dedicated command
- **Roll** γ, where γ is a reference to a past action
  - Undoes action pointed by γ, and all its consequences
  - Go back n steps not meaningful in a concurrent setting
- γ is a form of checkpoint
- This allows to make a computed result permanent
  - If there is no **roll** pointing back past a given action, then the action is never undone
- Still most programs are divergent

# External control

- Reversibility is controlled by something outside the program
- Again we have different possibilities
  - Controller processes
    [Iain Phillips, Irek Ulidowski, Shoji Yuen: A Reversible Process Calculus and the Modelling of the ERK Signalling Pathway. RC 2012]
  - Hierarchical component-based systems
  - Causal-consistent reversible debugger
- The other possibilities have not been fully explored yet

# Controller processes

- Two layered system
- A reversible slave process and a forward master process
- The slave process may execute only
  - Actions allowed by the master
  - In the direction allowed by the master
- The approach in the literature seems a bit ad hoc
- Used to model biological systems
- Allows for non causal-consistent reversibility

# Hierarchical component-based systems

- Systems featuring a hierarchy of components
- A generalization of the previous setting to multiple layers
- Each component controls the behavior of its children
  - Including the direction of their execution
- It needs information on the state of the children
  - E.g., each child should notify its errors

# Reversible debugger

- Standard debuggers provide commands to control the execution of the debugger program
  - Step
  - Run
  - ...
- A reversible debugger also provides the command "step back"
  - In a concurrent setting one should specify which thread should step back (or forward)
  - Some threads may be blocked and unable to step back (or forward)
- Commercial reversible debuggers exist

# Causal-consistent reversible debugger

- We want to exploit the causal information to help debugging concurrent applications
- We could provide a command like the **roll**
- Undo a given past action and all its consequences
- Currently work-in-progress
- Not yet clear which are the best commands to make available
- Not yet clear which is the desired debugging strategy
- More on this in the last lesson

# Semantic control

- Reversibility policy embedded in the language
- Again we have different possibilities
  - Prolog
  - State-space exploration via heuristics
  - Energy-based control
    [Giorgio Bacci, Vincent Danos, Ohad Kammar: On the Statistical Thermodynamics of Reversible Communicating Processes. CALCO 2011]

# Prolog backtracking

- Prolog tries to satisfy a given goal
- It explores deep-first the possible solutions
- When it reaches a dead end, it rollbacks and tries a different path
- The search is normally quite efficient
- Strongly relies on the programmer expertise

# State-space exploration via heuristics

- In general, there are different ways to explore a state space looking for a solution
- Strategy normally composed by a standard algorithm plus some heuristics driving it
- As before, if the algorithm reaches a dead end, it rollbacks and tries a different path
- Sample algorithm
  - count how many times each action has been done and undone
  - choose paths which have been tried more rarely

# Energy-based control

- We assume a world with a given amount of energy
- Forward and backward steps are taken subject to some probability
- The rates depend on the available amount of energy
- There is a lower bound on the amount of energy allowing to commit a forward computation in finite average time