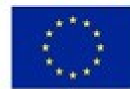
Causal-Consistent Reversible Debugging

Ivan Lanese
Focus research group
University of Bologna/INRIA

DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA



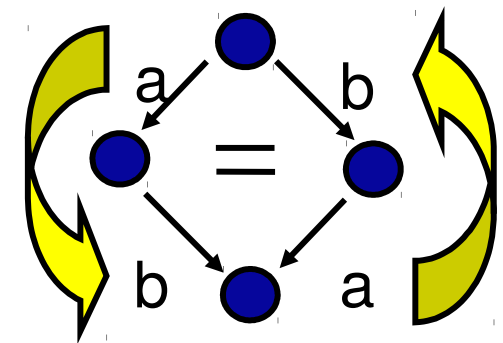
What is reversibility?

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

- In some areas systems are naturally reversible: biology, quantum computing, …
- In other areas making systems reversible can be useful: robotics, debugging, security, reliability, ...

Concurrent reversibility

- Reversibility in a sequential setting:
  recursively undo the last action
- In concurrent systems there is no uniquely defined last action
  - Many choices are possible
- We follow causal-consistent reversibility [Danos & Krivine, CONCUR 2004]
- Causal dependencies must be respected
  - First reverse the consequences, then the causes
- Independent actions are reversed independently



Reversibility for debugging

- Debugging amounts to find the wrong line of code (bug) causing a visible misbehavior
- The bug precedes and causes the misbehavior
  - Quite natural to use reversibility to go back from the misbehavior to the bug
- Sequential reversible debugging is well understood
  - Gdb (since 2009), Microsoft time-travel debugger, ...
- Concurrent reversible debugging not so developed
  - Most approaches just linearize the execution
  - Causal information is lost
- Can we use causal-consistent reversibility?

Causal-consistent debugging

- Introduced in [Giachino, Lanese & Mezzina, FASE 2014] inside ANR project REVER
- Allows one to explore a concurrent computation back and forward
  - Any action can be undone provided its consequences have been undone beforehand
- Which action to undo can be selected by the user or by a scheduler
- But we can do better

Debugging and causality

- Standard debugging procedure:
  1. Observing an unexpected behavior
  2. Finding in the code the instruction that caused it
  3. Correcting the instruction
- Causal-consistent reversibility includes lot of causal information
- This information can be used to drive step 2 above
- Debugging strategy: follow causality links backward from the misbehavior to the bug
- Which primitives do we need to enable such a strategy?
- We introduced the roll operator

Causal-consistent debugging: roll

- The roll operator allows one to undo a selected past action, including all and only its consequences
- Minimal set of undos needed to undo the selected action in a causal-consistent way
- Many interfaces for it:
  - N actions in a given process
  - Last assignment to a given variable
  - Send of a given message
- Dual approach for forward execution: redo a future action (from a log) including all and only its causes

Causal-consistent roll at work

- The programmer executes the program and finds some unexpected behavior
- The roll allows him to find automatically the instruction that immediately caused the misbehavior
- Two possibilities:
  - The found instruction is wrong: bug found
  - The found instruction gets wrong data from previous instructions: iterate
- One can explore the tree of causes, navigating from one process to the other

CauDEr

- Causal-consistent Debugger for Erlang
- Applies the approach outlined above to Erlang
  - Functional and concurrent language
  - Used in mainstream applications such as some versions of Facebook chat
- Currently CauDEr moving to version 2 (almost there)
  - From Core Erlang to Erlang
  - New interface
  - https://github.com/mistupv/cauder-v2
- Mainly a collaboration between FOCUS and Universitat Politècnica de València

CauDEr v2

CauDEr

File Edit View Run Help

Code

```
20              {read,Pid} -> Pid!Val end,
21      varManager(Val).
22
23  incrementer(MePid,XPid) ->
24      MePid!{request,self()},
25      receive answer ->
26              XPid!{read,self()},
27              receive X ->
28                      XPid!{write,X+1},
29                      MePid!{release} end end.
30
```

98 ! {read, 99}

Process Info

Bindings

Name	Value
MePid	97
XPid	98

Stack

receive
meViolation:incrementer/2

Log

send(3)
rec(4)
send(5)
send(6)

History

rec(answer,1)
send({request,99},0)

Actions

Process

PID: 99 - meViolation:incrementer/2

Manual | Automatic | Replay | Rollback

Steps: 1 — Roll steps

PID: — Roll spawn

Msg. Uid: 3 — Roll send

Msg. Uid: — Roll receive

Var. — Roll variable

System Info

Mail

Dest.	Value	UID
97	{request,100}	2

Trace | Roll Log

Roll send from Proc. 99 of {release} to Proc. 97 (6)
Roll send from Proc. 97 of answer to Proc. 100 (7)

Rolled back sending of message with UID: 3 — Ln 26, Col 1 — Alive 4, Dead 1

Future directions

- Extending the supported fragment of the language
  - Currently functional and concurrent features
  - Error handling and distribution are under development
- Refine the causal approach
  - What if analysis and causal compression
- Improving the efficiency
  - Memory and time overhead due to history information

Causal-consistent reversible debugging

Causal-consistent reversibility, Danos & Krivine,CONCUR 2004
31/8/04

CauDEr, Lanese, Nishida, Palacios & Vidal, FLOPS 2018
9/5/18

Causal-consistent debugging, Giachino, Lanese & Mezzina, FASE 2014
5/4/14

Today

ANR DCore
54 months
31/3/2019 - 29/9/2023

ANR REVER
48 months
1/12/2011 - 30/11/2015

COST Action on Reversible Computation
46 months
1/7/2015 - 30/4/2019

Questions?