SOFTWARE DEVELOPMENT & REVERSIBILITY: OPEN PROBLEMS

Claudio Antares Mezzina
(with special suggestions from Lanese & Ulidowski & Tuosto)
IMT School for Advanced Studies Lucca
WG1 Meeting, Cyprus


NOVITÀ
OMBRELLI
REVERSIBILI
€ 12.00
BASKET
FUTSAL
1
2

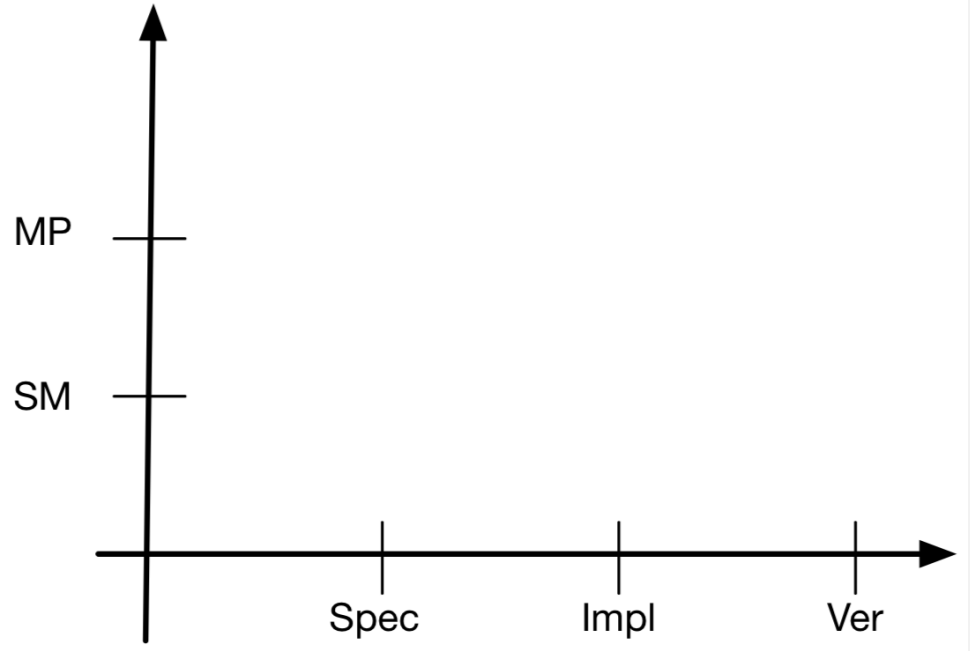
SOFTWARE DEVELOPMENT

- According to Software Engineering we can distinguish three main phases during software development
  - **Specification**
  - **Implementation**
  - **Validation / Verification**

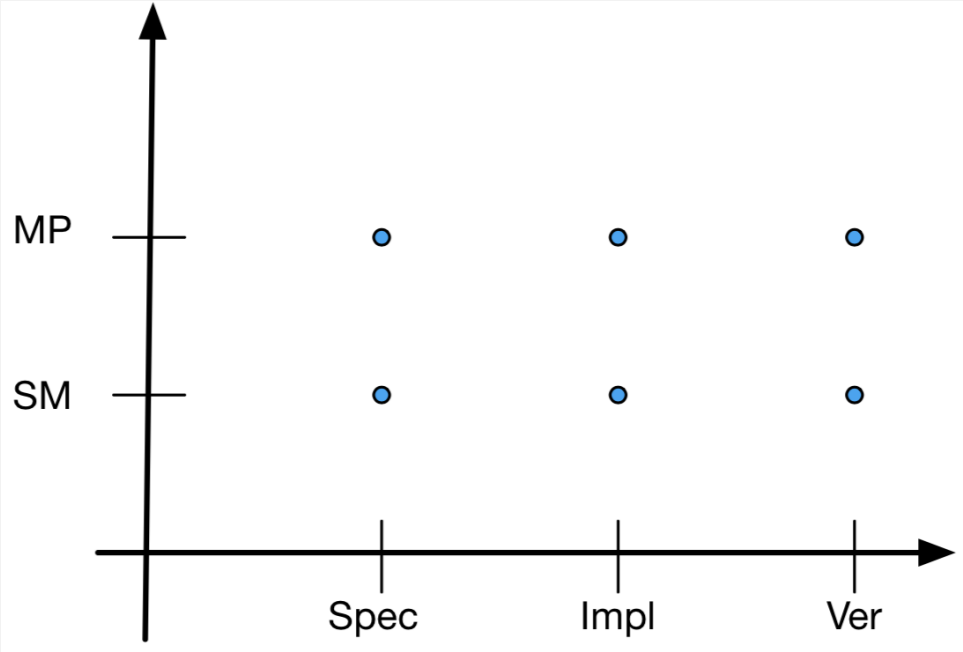
SEQUENTIAL VS CONCURRENT

- In this talk we will just focus on Concurrent Programs
- There exists two concurrency models
  - Shared Memory **SM**
  - Message Passing **MP**

PROBLEM SPECTRUM



PROBLEM SPECTRUM



CONCURRENT

- **Specification**
  - **There exist formalisms which are general enough to work with both (Spec, MP) and (Spec, SM)**
    - Reversible Prime Event Structures / (Phillips & Ulidowski)
    - Rigid Families (Cristescu & Krivine & Varacca)
    - Reversible CCS / pi-calculus (Krivine et al., Lanese et al., Phillips et al.)

CONCURRENT: TOOLBOX

- **(Spec, MP)**
  - Reversible Contracts (Barbanera & De' Liguoro &Lanese)
    - Binary, limited to choices
  - Reversible Global Graphs (Mezzina & Tuosto)
    - Multiparty, choices and loops + conditions

CONCURRENT: TOOLBOX

- **(Spec, MP)**
  - Reversible (Multi Party -) Session Types
    - Tiezzi & Yoshida types are not used at all
    - Castellani et al multiparty, limited to choices
    - Mezzina & Perez multiparty, fully reversible

CONCURRENT

- **(Spec, SM)**
  - **Based on shared memory**
    - Some initial ideas on reversible SOS with store
    - Compared with the MP scenario, we are lacking *alternatives*

IMPLEMENTATION

- **(Impl, MP)**
  - Reversible Erlang (Lanese & Nishida & Palacios & Vidal)
  - Reversible Communicating Machines (**ongoing** Mezzina & Tuosto & Ulidowski)
  - Actor model + Checkpoint (Transactors)

CONCURRENT SETTING: *IMPLEMENTATION*

- **(Impl, SM)**
  - Reversing Parallel Programs (Hoey & Ulidowski & Yuen)
  - Reversible Tuple Spaces (Giachino & Lanese & Mezzina & Tiezzi)
  - Compared with the MP scenario, we are lacking *alternatives*

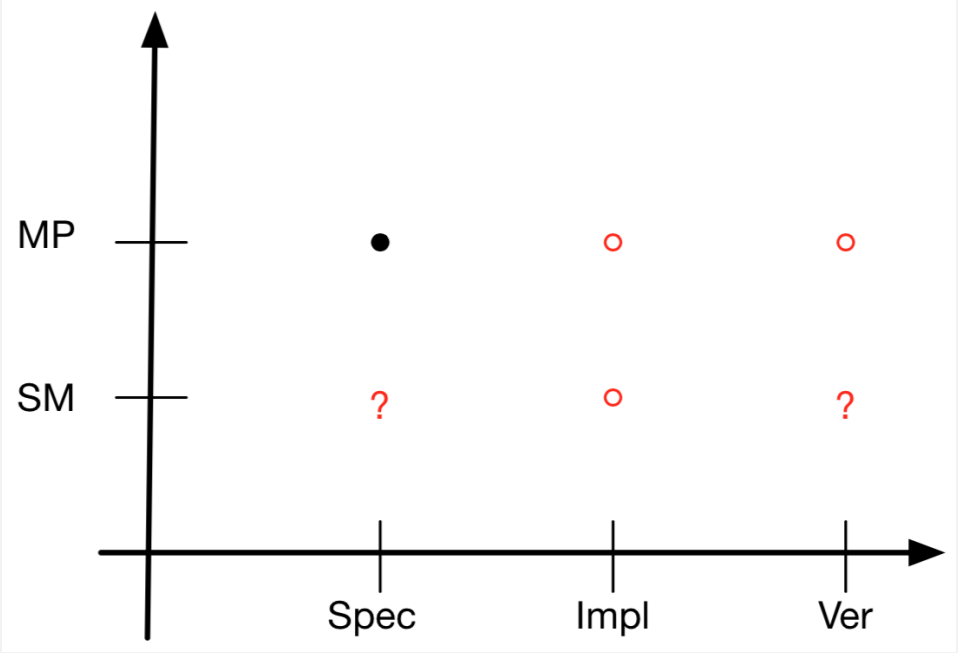
VERIFICATION

- **There exist formalisms which are general enough to work with both (Ver, MP) and (Ver, SM)**
- **Equivalences for Reversibility**
  - **Bisimulations:** few works but still far away from something concrete
    - See Iain's slides for the Training School
  - **Testing:** some preliminary ideas
    - A safety and liveness theory for total reversibility **(**Mezzina & Koutavas**)**
    - More on Ivan's talk from last WG1 meeting
- **Logics for reversible systems**
  - Event Identifier Logic (Phillips & Ulidowski)

VERIFICATION

- **(Ver, MP)**
  - **Debugging**
    - CC Reversible Erlang Debugger (Lanese & Nishida & Palacios & Vidal)
      - Based on fully reversible semantics
- **(Ver, SM)**
  - **Debugging**
    - mOz debugger for a toy language (Giachino & Lanese & Mezzina)
    - No real CC Reversible Deb for Shared Memory

SPECTRUM



RESEARCH GOAL

- If we were to apply any *meaningful* combination of the above mentioned approaches to software development

Do we get a full-fledged framework for software development?

EXAMPLE

- Specification: Prime **Event Structrure**
- Implementation: **Reversible Parallel programs**
- Verification: **Event Identifiers Logic**

OPEN QUESTIONS I

- Specification: **Reversible Prime Event Structrure**
- Implementation: **Reversible Parallel programs**
- Verification: **Event Identifiers Logic**

- **How can we map a program execution into a RPES?**
- **How can we extend EIL to query such structures?**
  - **e.g. what is the minimal cause of a certain event ?**

OPEN QUESTIONS 2

- Starting from a program specification / trace
  - How can we generate *reversible* tests?
  - How can tests be transformed into logic formulae?

OPEN QUESTION 3

- **Record / Replay**
  - No techniques for CC record-replay and trace-compression
  - How can we record an execution and compress it?
    - we can record a faulty execution of a *beta* program and debug it later on
    - from the faulty trace we can automatically generate tests to see whether the bug is solved
  - When two traces are equivalent?
  - How we can compress trace to save time (while recording) and space?

OPEN QUESTION 4

- So far reversible debuggers use a fully reversible semantics of the source language
  - You have to *hack* the semantics of the VM/interpreter in order to embedd
    - History logging
    - Backward execution

OPEN QUESTION 4

- Can’t we exploit built in mechanisms of the soure language?
- Can we build a debugging facility on top of the Erlang supervision model?
  - Ability to *start* / *stop*/ *restart* actors
  - Ability to query the status of actors

OPEN QUESTIONS
5

Any other questions?