ALMArie CURIE 2021– SupER- funded
on D.M. 737/2021 resources-funded by
European Union – “NextGenerationEU"

A general approach to derive causally-consistent reversible semantics

Ivan Lanese Doriana Medić Giovanni Fabbretti Jean-Bernard Stefani

Work partially supported by French ANR project DCore ANR-18-CE25-0007 and
by ALMArie CURIE 2021 project J45F21001470005

OPCT, June 26-30, 2023

Material from CONCUR 2020 and ICFEM 2022

Reversible computing

- already discussed in previous talks by Claudio M. and Irek, but in case you were distracted . . .
- allows one to execute programs not only forwards, but also backwards
- applied in low-power computing, biochemical modelling, simulation, robotics, debugging, etc.
- sequential systems - forward actions are undone in reverse order
- concurrent systems - identifying the last action is not immediate

Causally-consistent reversibility

- proposed by Danos & Krivine in their CONCUR 2004 paper
  - the paper has won the CONCUR 2023 test-of-time award
- any action can be reversed, provided that all its consequences have been reversed beforehand
- dependent actions must be reversed in reverse order, concurrent actions can be reversed in any order
- mainstream approach to reversibility in concurrent systems, albeit other approaches exist

[1] V. Danos and J. Krivine: Reversible communicating systems. In: CONCUR 2004.

Working on causally-consistent reversibility

- Claudio M. showed what a reversible timed process calculus looks like
- . . . and causally-consistent extensions of many formalisms (CCS, pi, occurrence nets, event structures, Erlang, . . .) exist
- . . . but most of the constructions are ad hoc
- can we provide a general technique?

Our answer

- partially positive answer
- positive for systems with reduction semantics satisfying suitable conditions and causality based on resources produced and consumed
- a causally-consistent reversible semantics can be automatically produced
- it satisfies all the expected properties (cf. Irek's talk)
- two case studies: (asynchronous) Higher-Order π-calculus and Core Erlang (also features not previously considered in the literature)
- approach implemented in Maude

The requirements on the forward model

The forward model needs to be structured in two levels:

The requirements on the forward model

The forward model needs to be structured in two levels:

- The lower level is composed of entities (processes, messages, resources) ranged over by P, Q.
  - No restrictions on the syntax of the lower level.

The requirements on the forward model

The forward model needs to be structured in two levels:

- The lower level is composed of entities (processes, messages, resources) ranged over by P, Q.
  - No restrictions on the syntax of the lower level.

- The higher level is composed of the following operators:

$$N ::= P \ | \ op_n(N_1, \ldots, N_n) \ | \ 0$$

  where

  - $op_n(N_1, \ldots, N_n)$ stands for a family of operators;
  - parallel composition, $N_1 \mid N_2$, is assumed among the operators;
  - 0 represents the empty system.

Running example: the HOπ-calculus

- The syntax of the HOπ-calculus is:

$$P ::= a\langle P\rangle \mid a(X) \triangleright P \mid \ (P_1 \mid P_2) \ \mid \nu a\,(P) \mid X \mid 0$$

  - we separate entities from systems;

  - an entity Q is any HOπ process whose topmost operator is neither a parallel composition nor a restriction nor 0.

Running example: the HOπ-calculus

- The syntax of the HOπ-calculus is:

$$P ::= a\langle P\rangle \mid a(X) \triangleright P \mid \ (P_1 \mid P_2) \ \mid \nu a\,(P) \mid X \mid 0$$

  - we separate entities from systems;
  - an entity Q is any HOπ process whose topmost operator is neither a parallel composition nor a restriction nor 0.

- The syntax of systems is defined as:

$$N := Q \mid \ (N_1 \mid N_2) \ \mid \nu a\,(N) \mid 0$$

where:

- operators $\mid$ and 0 are required by our framework;
- restriction is an infinite family of unary operators with one instance for each name a.

Structural congruence

- A generic system can be represented as a term

$$T[P_1, \ldots, P_n]$$

where $T[\bullet_1, \ldots, \bullet_n]$ is a context with n numbered holes.

- T is built from composition operators, possibly including parallel composition and 0.

Structural congruence

- A generic system can be represented as a term

$$T[P_1, \dots, P_n]$$

where $T[\bullet_1, \dots, \bullet_n]$ is a context with n numbered holes.

- T is built from composition operators, possibly including parallel composition and 0.

- Structural congruence is specified by axioms of the form:

$$T[P_1, \dots, P_n] \equiv T'[P'_1, \dots, P'_n]$$

closed under contexts, reflexivity, symmetry and transitivity.

Structural congruence

- A generic system can be represented as a term

$$T[P_1, \ldots, P_n]$$

where $T[\bullet_1, \ldots, \bullet_n]$ is a context with n numbered holes.

- T is built from composition operators, possibly including parallel composition and 0.

- Structural congruence is specified by axioms of the form:

$$T[P_1, \ldots, P_n] \equiv T'[P'_1, \ldots, P'_n]$$

closed under contexts, reflexivity, symmetry and transitivity.

- **Example:** A sample HOπ structural rule is:

$$(\textsc{ParC}) \;\; P \mid Q \equiv Q \mid P$$

- it exploits contexts of the form $\bullet_1 \mid \bullet_2$ and $\bullet_2 \mid \bullet_1$

The reduction semantics of the forward model

$$(\textsc{Scm-Act})\ \frac{}{P_1 \mid \ldots \mid P_n \rightarrowtail T[Q_1, \ldots, Q_m]}$$

$$(\textsc{Eqv})\ \frac{N \equiv N' \quad N \rightarrowtail N_1 \quad N_1 \equiv N_1'}{N' \rightarrowtail N_1'}$$

$$(\textsc{Scm-Opn})\ \frac{N_i \rightarrowtail N_i'}{op_n(N_0, \ldots, N_i, \ldots, N_n) \rightarrowtail op_n(N_0, \ldots, N_i', \ldots, N_n)}$$

$$(\textsc{Par})\ \frac{N \rightarrowtail N'}{N \mid N_1 \rightarrowtail N' \mid N_1}$$

Running example: the communication rule of HOπ

- The communication rule $(\textsc{Act})$ of HOπ is defined as:

$$(\textsc{Act})\ \frac{}{a\langle Q\rangle \mid a(X) \triangleright P \rightarrowtail P\{Q/X\}}$$

Running example: the communication rule of HOπ

- The communication rule $(\textsc{Act})$ of HOπ is defined as:

$$(\textsc{Act})\ \frac{}{a\langle Q\rangle \mid a(X) \triangleright P \rightarrowtail P\{Q/X\}}$$

- Gives rise to an infinite number of instances
- The number of entities in the resulting process may vary:

$$a\langle b\langle P\rangle \mid b(Y) \triangleright Y \mid c\langle Q\rangle\rangle \mid a(X) \triangleright X \rightarrowtail b\langle P\rangle \mid b(Y) \triangleright Y \mid c\langle Q\rangle$$

- the resulting process has three entities $b\langle P\rangle$, $b(Y) \triangleright Y$ and $c\langle Q\rangle$, composed using a context $T = \bullet_1 \mid \bullet_2 \mid \bullet_3$.

Definition of the reversible configuration

- The syntax of reversible configurations R is:

$$R ::= k : P \mid op_n(R_1, \dots, R_n) \mid 0 \mid [R; C] \qquad C ::= T[k_1 : \bullet_1, \dots, k_m : \bullet_m]$$

where:

- k denotes a key identifying each entity of a system;
- op_n are the same as in the forward system;
- T is a context composed of operators op_n and 0;
- $\bullet_i$ are numbered holes, to be filled by the processes with keys k_i;
- $[R; C]$ is a memory (R is the configuration which gave rise to the forward step and C is the context of the resulting configuration).

Running example: syntax of reversible HOπ

- The syntax of the reversible HOπ-calculus is defined as:

$$R ::= k : Q \mid (R_1 \mid R_2) \mid \nu a\, (R) \mid 0 \mid [R; C]$$

where Q are the entities as in the underlying calculus.

Structural congruence in the reversible calculus: naive approach

- For each axiom:

$$T[P_1, \ldots, P_n] \equiv T'[P'_1, \ldots, P'_n]$$

let us define a corresponding axiom:

$$T[k_1 : P_1, \ldots, k_n : P_n] \equiv_k T'[k_1 : P'_1, \ldots, k_n : P'_n]$$

- entities are labelled with keys and keys on both sides are the same.
- NOT WORKING, since it is not able to deal with memories
  - E.g., we cannot apply commutativity of parallel composition to swap memories

Structural congruence in the reversible calculus: refined approach

$$T[k_1 : P_1, \ldots, k_n : P_n] \equiv_k T'[k_1 : P'_1, \ldots, k_n : P'_n]$$

We apply the axioms to a flat representation of the configuration

$$R_1 \equiv_c R_2 \text{ iff there are } R'_1, R'_2, S \text{ such that}$$
$$\mathtt{proj}(R_1) = (R'_1, S) \wedge R'_1 \equiv_k R'_2 \wedge \mathtt{proj}(R_2) = (R'_2, S)$$

where:

- $\mathtt{proj}([R; C]) = (R, \{(\mathtt{key}(R), C)\})$
- projection is defined homomorphically on the other operators;
- old configurations R are freed in the current configuration;
- $\{(\mathtt{key}(R), C)\}$ remembers information on what was in the memories.

Applying the refined structural congruence

$$R = \nu a\, (j_1 : P_1 \mid j_2 : P_2 \mid [k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X;\, T[j_1 : \bullet_1, j_2 : \bullet_2]])$$

$$\begin{aligned}
\mathtt{proj}(R) \quad &= \quad (\nu a\, (j_1 : P_1 \mid j_2 : P_2 \mid k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X), \\
&\qquad \{(\{k_1, k_2\}, T[j_1 : \bullet_1, j_2 : \bullet_2])\})
\end{aligned}$$

$$\begin{aligned}
\mathtt{proj}(R') \quad &= \quad (\nu b\, (j_1 : P_1 \mid j_2 : P_2 \mid k_1 : b\langle P_1 \mid P_2\rangle \mid k_2 : b(X) \triangleright X), \\
&\qquad \{(\{k_1, k_2\}, T[j_1 : \bullet_1, j_2 : \bullet_2])\})
\end{aligned}$$

$$R' = \nu b\, (j_1 : P_1 \mid j_2 : P_2 \mid [k_1 : b\langle P_1 \mid P_2\rangle \mid k_2 : b(X) \triangleright X;\, T[j_1 : \bullet_1, j_2 : \bullet_2]])$$

- each structural axiom needs to satisfy some coherence conditions;
- essentially content of the memories should not enable new transitions;
- satisfied for most structural axioms (e.g., HOπ axioms).

The reversible semantics

- The forward rules of the uncontrolled reversible semantics are:

$$(\textsc{F-Scm-Act})\ \frac{P_1 \mid \ldots \mid P_n \rightarrowtail T[Q_1, \ldots, Q_m] \qquad j_1, \ldots, j_m \text{ are fresh keys}}{\begin{array}{l} k_1 : P_1 \mid \ldots \mid k_n : P_n \twoheadrightarrow T[j_1 : Q_1, \ldots, j_m : Q_m] \mid \\ \qquad\qquad\qquad\qquad [k_1 : P_1 \mid \ldots \mid k_n : P_n; T[j_1 : \bullet_1, \ldots, j_m : \bullet_m]] \end{array}}$$

$$(\textsc{F-Scm-Opn})\ \frac{R_i \twoheadrightarrow R_i' \quad (\mathtt{key}(R_i') \setminus \mathtt{key}(R_i)) \cap (\mathtt{key}(R_0, \ldots, R_{i-1}, R_{i+1}, \ldots, R_n) = \emptyset}{op_n(R_0, \ldots, R_i, \ldots, R_n) \twoheadrightarrow op_n(R_0, \ldots, R_i', \ldots, R_n)}$$

$$(\textsc{F-Eqv})\ \frac{R \equiv_c R' \quad R \twoheadrightarrow R_1 \quad R_1 \equiv_c R_1'}{R' \twoheadrightarrow R_1'}$$

The reversible semantics

- The forward rules of the uncontrolled reversible semantics are:

$$\text{(F-SCM-ACT)}\ \frac{P_1 \mid \ldots \mid P_n \rightarrowtail T[Q_1, \ldots, Q_m] \qquad j_1, \ldots, j_m \text{ are fresh keys}}{k_1 : P_1 \mid \ldots \mid k_n : P_n \twoheadrightarrow T[j_1 : Q_1, \ldots, j_m : Q_m] \mid [k_1 : P_1 \mid \ldots \mid k_n : P_n; T[j_1 : \bullet_1, \ldots, j_m : \bullet_m]]}$$

$$\text{(F-SCM-OPN)}\ \frac{R_i \twoheadrightarrow R_i' \quad (\mathtt{key}(R_i') \setminus \mathtt{key}(R_i)) \cap (\mathtt{key}(R_0, \ldots, R_{i-1}, R_{i+1}, \ldots, R_n) = \emptyset}{op_n(R_0, \ldots, R_i, \ldots, R_n) \twoheadrightarrow op_n(R_0, \ldots, R_i', \ldots, R_n)}$$

$$\text{(F-EQV)}\ \frac{R \equiv_c R' \quad R \twoheadrightarrow R_1 \quad R_1 \equiv_c R_1'}{R' \twoheadrightarrow R_1'}$$

- The backward rules are symmetric w.r.t. the forward ones.

Running example: sample HOπ reduction

- consider system $R = k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X$.

Running example: sample HOπ reduction

- consider system $R = k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X$.

- forward step:

$$k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X \twoheadrightarrow$$

Running example: sample HOπ reduction

- consider system $R = k_1 : a\langle P_1 \mid P_2 \rangle \mid k_2 : a(X) \triangleright X$.

- forward step:

$$k_1 : a\langle P_1 \mid P_2 \rangle \mid k_2 : a(X) \triangleright X \twoheadrightarrow j_1 : P_1 \mid j_2 : P_2 \mid$$

Running example: sample HOπ reduction

- consider system $R = k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X$.

- forward step:

$$k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X \twoheadrightarrow j_1 : P_1 \mid j_2 : P_2 \mid [R; T[j_1 : \bullet_1, j_2 : \bullet_2]]$$

Running example: sample HOπ reduction

- consider system $R = k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X$.

- forward step:

$$k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X \twoheadrightarrow j_1 : P_1 \mid j_2 : P_2 \mid [R; T[j_1 : \bullet_1, j_2 : \bullet_2]]$$

where $T = \bullet_1 \mid \bullet_2$.

Running example: sample HOπ reduction

- consider system $R = k_1 : a\langle P_1 \mid P_2 \rangle \mid k_2 : a(X) \triangleright X$.

- forward step:

$$k_1 : a\langle P_1 \mid P_2 \rangle \mid k_2 : a(X) \triangleright X \twoheadrightarrow j_1 : P_1 \mid j_2 : P_2 \mid [R; T[j_1 : \bullet_1, j_2 : \bullet_2]]$$

where $T = \bullet_1 \mid \bullet_2$.

- backward step:

$$j_1 : P_1 \mid j_2 : P_2 \mid [R; T[j_1 : \bullet_1, j_2 : \bullet_2]] \rightsquigarrow$$

Running example: sample HOπ reduction

- consider system $R = k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X$.

- forward step:

$$k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X \twoheadrightarrow j_1 : P_1 \mid j_2 : P_2 \mid [R; T[j_1 : \bullet_1, j_2 : \bullet_2]]$$

where $T = \bullet_1 \mid \bullet_2$.

- backward step:

$$j_1 : P_1 \mid j_2 : P_2 \mid [R; T[j_1 : \bullet_1, j_2 : \bullet_2]] \rightsquigarrow k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X$$

Concurrent transitions

- to discuss the properties of our semantics we need a notion of concurrency (or, dually, causality)
- we extract the definition of concurrency from the reversible syntax
- transitions t of a system R are defined as:

$$t : R \xrightarrow{\mu} R'$$

where μ is the memory created by the transition, if it is forward, or consumed by it, if it is backward

- the function $\mathtt{key}(\cdot)$ computes the set of keys of a given system.

Definition (Concurrent transitions)

Two coinitial transitions $t' : R \xrightarrow{\mu'} R'$ and $t'' : R \xrightarrow{\mu''} R''$ are **concurrent** if $\mathtt{key}(\mu') \cap \mathtt{key}(\mu'') = \emptyset$.

Properties

◦ our framework fits the axiomatic meta-model described in Irek's talk [1], hence it enjoys a number of relevant properties, such as:

[1] I. Lanese, I. C. C. Phillips and I. Ulidowski: An axiomatic approach to reversible computation. In FOSSACS 2020.

Properties

◦ our framework fits the axiomatic meta-model described in Irek's talk [1], hence it enjoys a number of relevant properties, such as:

- Loop Lemma - every action can be undone;
- Parabolic Lemma - each reversible computation can be rearranged as a backward computation, followed by a forward one;
- Causal Consistency - the correct history and causality information is stored;
- Causal Safety - an action cannot be reversed until all actions caused by it have been reversed;
- Causal Liveness - actions can be reversed in any order consistent with Causal Safety.

[1] I. Lanese, I. C. C. Phillips and I. Ulidowski: An axiomatic approach to reversible computation. In FOSSACS 2020.

Case study: Core Erlang

- Core Erlang is Erlang stripped of syntactic sugar (it was used as intermediate step in Erlang compilation)

Case study: Core Erlang

◦ Core Erlang is Erlang stripped of syntactic sugar (it was used as intermediate step in Erlang compilation)

◦ A Core Erlang system [1] is defined as:

$$E := \langle p, \theta, e \rangle \mid$$

where

- $\langle p, \theta, e \rangle$ - a process with a pid p evaluating expression e in environment θ;

Case study: Core Erlang

◦ Core Erlang is Erlang stripped of syntactic sugar (it was used as intermediate step in Erlang compilation)

◦ A Core Erlang system [1] is defined as:

$$E := \langle p, \theta, e \rangle \mid (p, p', v) \mid$$

where

- $\langle p, \theta, e \rangle$ - a process with a pid p evaluating expression e in environment θ;
- (p, p', v) - a message carrying value v sent by the process with pid p to the one with pid p'.

Case study: Core Erlang

- Core Erlang is Erlang stripped of syntactic sugar (it was used as intermediate step in Erlang compilation)
- A Core Erlang system [1] is defined as:

$$E := \langle p, \theta, e \rangle \mid (p, p', v) \mid \ (E_1 \mid E_2)$$

where

- $\langle p, \theta, e \rangle$ - a process with a pid p evaluating expression e in environment θ;
- (p, p', v) - a message carrying value v sent by the process with pid p to the one with pid p'.

Case study: Core Erlang

◦ Core Erlang is Erlang stripped of syntactic sugar (it was used as intermediate step in Erlang compilation)

◦ A Core Erlang system [1] is defined as:

$$E := \langle p, \theta, e \rangle \mid (p, p', v) \mid \ (E_1 \mid E_2)$$

where

- $\langle p, \theta, e \rangle$ - a process with a pid p evaluating expression e in environment θ;
- (p, p', v) - a message carrying value v sent by the process with pid p to the one with pid p'.

◦ From our approach we get that a reversible Core Erlang configuration is defined as:

$$R ::= k : \langle p, \theta, e \rangle \mid k : (p, p', v) \ \mid \ (R_1 \mid R_2) \mid [R; C]$$

Sample Core Erlang reduction

- Rule (Send) of Core Erlang semantics:

$$\langle p, \theta, p'!5 \rangle \hookrightarrow \langle p, \theta, 5 \rangle \mid (p, p', 5)$$

Sample Core Erlang reduction

- Rule $(\textsc{Send})$ of Core Erlang semantics:

$$\langle p, \theta, p'!5\rangle \hookrightarrow \langle p, \theta, 5\rangle \mid (p, p', 5)$$

- Forward rule $(\text{F-}\textsc{Send})$ of the reversible semantics for Erlang:

$k : \langle p, \theta, p'!5\rangle \twoheadrightarrow$

Sample Core Erlang reduction

- Rule $(\textsc{Send})$ of Core Erlang semantics:

$$\langle p, \theta, p'!5\rangle \hookrightarrow \langle p, \theta, 5\rangle \mid (p, p', 5)$$

- Forward rule $(\textsc{F-Send})$ of the reversible semantics for Erlang:

$$k : \langle p, \theta, p'!5\rangle \twoheadrightarrow k_1 : \langle p, \theta, 5\rangle \mid k_2 : (p, p', 5) \mid$$

Sample Core Erlang reduction

- Rule $(\textsc{Send})$ of Core Erlang semantics:

$$\langle p, \theta, p'!5\rangle \hookrightarrow \langle p, \theta, 5\rangle \mid (p, p', 5)$$

- Forward rule $(\textsc{F-Send})$ of the reversible semantics for Erlang:

$$k :\langle p, \theta, p'!5\rangle \twoheadrightarrow k_1 :\langle p, \theta, 5\rangle \mid k_2 :(p, p', 5) \mid [k : \langle p, \theta, p'!5\rangle; k_1 : \bullet_1 \mid k_2 : \bullet_2]$$

Sample Core Erlang reduction

- Rule $(\textsc{Send})$ of Core Erlang semantics:

$$\langle p, \theta, p'!5\rangle \hookrightarrow \langle p, \theta, 5\rangle \mid (p, p', 5)$$

- Forward rule $(\text{F-}\textsc{Send})$ of the reversible semantics for Erlang:

$$k : \langle p, \theta, p'!5\rangle \twoheadrightarrow k_1 : \langle p, \theta, 5\rangle \mid k_2 : (p, p', 5) \mid [k : \langle p, \theta, p'!5\rangle; k_1 : \bullet_1 \mid k_2 : \bullet_2]$$

- Backward rule $(\text{B-}\textsc{Send})$ of the reversible semantics for Erlang:

$$k_1 : \langle p, \theta, 5\rangle \mid k_2 : (p, p', 5) \mid [k : \langle p, \theta, p'!5\rangle; k_1 : \bullet_1 \mid k_2 : \bullet_2] \rightsquigarrow$$

Sample Core Erlang reduction

- Rule $(\textsc{Send})$ of Core Erlang semantics:

$$\langle p, \theta, p'!5 \rangle \hookrightarrow \langle p, \theta, 5 \rangle \mid (p, p', 5)$$

- Forward rule $(\text{F-}\textsc{Send})$ of the reversible semantics for Erlang:

$$k : \langle p, \theta, p'!5 \rangle \twoheadrightarrow k_1 : \langle p, \theta, 5 \rangle \mid k_2 : (p, p', 5) \mid [k : \langle p, \theta, p'!5 \rangle; k_1 : \bullet_1 \mid k_2 : \bullet_2]$$

- Backward rule $(\text{B-}\textsc{Send})$ of the reversible semantics for Erlang:

$$k_1 : \langle p, \theta, 5 \rangle \mid k_2 : (p, p', 5) \mid [k : \langle p, \theta, p'!5 \rangle; k_1 : \bullet_1 \mid k_2 : \bullet_2] \rightsquigarrow k : \langle p, \theta, p'!5 \rangle$$

Correspondence between two reversible semantics for Erlang

- Our semantics is equivalent to the one in [1]:

Theorem (Causal correspondence)

Two coinitial transitions t_1 and t_2 of our reversible Core Erlang semantics are concurrent according to [1] iff they are concurrent according to our definition.

Theorem (Bisimulation)

The reversible semantics of Core Erlang in [1] and our reversible semantics of Core Erlang are strong back and forth barbed bisimilar.

- ... but we can deal with additional constructs, e.g., for error propagation.

[1] I. Lanese, A. Palacios, G. Vidal: Causal-Consistent Replay Reversible Semantics for Message Passing Concurrent Programs. Fundam. Informaticae 178(3), 2021

Maude implementation

- we implemented a Maude program which takes as input a Maude semantics with a suitable structure and computes the corresponding reversible semantics;
- main conceptual issue: how to give a finite representation for the infinite sets of rules;
- rules are generated from schemas with side conditions;
- the same side conditions are used in the forward semantics, no side conditions are needed in the backward one.

Maude: Erlang send

```
crl [sys-send] :
    < P | exp: EXSEQ, env-stack: ENV, ASET > =>
    < P | exp: EXSEQ', env-stack: ENV', ASET > ||
    < sender: P, receiver: DEST, payload: GVALUE >
    if < DEST ! GVALUE, ENV', EXSEQ' > :=
       < req-gen, ENV, EXSEQ > .
```

Maude: forward and backward Erlang send

```
crl [fwd sys-send]:
    < P | ASET, exp: EXSEQ, env-stack: ENV > * key(L) =>
    < sender: P, receiver: DEST, payload: GVALUE > * key(0 L) ||
    < P | exp: EXSEQ', env-stack: ENV', ASET > * key(1 L) ||
    [< P | ASET, exp: EXSEQ, env-stack: ENV > * key(L) ;
      @: key(0 L) || @: key(1 L)]
    if < DEST ! GVALUE, ENV', EXSEQ' > :=
       < req-gen, ENV, EXSEQ > .
```

Maude: forward and backward Erlang send

```
crl [fwd sys-send]:
    < P | ASET, exp: EXSEQ, env-stack: ENV > * key(L) =>
    < sender: P, receiver: DEST, payload: GVALUE > * key(0 L) ||
    < P | exp: EXSEQ', env-stack: ENV', ASET > * key(1 L) ||
    [< P | ASET, exp: EXSEQ, env-stack: ENV > * key(L) ;
      @: key(0 L) || @: key(1 L)]
    if < DEST ! GVALUE, ENV', EXSEQ' > :=
       < req-gen, ENV, EXSEQ > .

crl [bwd sys-send]:
    < sender: P, receiver: DEST, payload: GVALUE > * key(0 L) ||
    < P | exp: EXSEQ', env-stack: ENV', ASET > * key(1 L) ||
    [< P | ASET, exp: EXSEQ, env-stack: ENV > * key(L) ;
      @: key(0 L) || @: key(1 L)] =>
    < P | ASET, exp: EXSEQ, env-stack: ENV > * key(L)
```

Conclusion and open problems

- to summarise:
  - a fully automatic method to extend a given forward model to a causally-consistent reversible one;
  - for our case studies, the obtained reversible semantics are equivalent to the ones in the literature;
  - the approach has been implemented in Maude.
- open problems:
  - dealing with control mechanisms such as irreversible actions or rollback operators;
  - extend the approach to handle other concurrency models
    - to deal with (atomic) imperative variables we need to be able to read a resource without consuming it;
    - what about general data structures such as sets or user-defined types?

Thank you for attention ☺

Questions?