A general approach to derive uncontrolled reversible semantics

DCore meeting Paris, 2-3 March 2020

Ivan Lanese and Doriana Medić

February 27, 2020

DCore

Introduction

- In this work we are interested in obtaining a general approach for transforming a given forward semantics into the reversible one.
- In particular, our aim is:
  - to obtain a general approach applicable on different models. We use as the case studies Core Erlang and Higher-Order π-calculus (the result holds for π-calculus and CCS, as well);
  - to show that the general framework enjoys the properties as Loop Lemma, Square Lemma and Causal Consistency. In that way in the newly obtained reversible semantics, mentioned properties shall hold by construction;
  - to extend the reversible semantics of Core Erlang to support constructs for error handling mechanism based on links

The forward model

The forward model needs to satisfy the following requirements.

- The syntax of a system is given in two levels. The basic level is represented by the entities P, Q, which could be processes, messages, etc.

- The high-level syntax of the system is defined as:

$$N ::= P \mid (N_1 \mid N_2) \mid opn(N_1, \ldots, N_n)$$

  where

  - $N_1 \mid N_2$ symbolises a parallel composition of systems N_1 and N_2;
  - $opn(N_1, \ldots, N_n)$ represents the n-ary operations applied on systems $N_1, \ldots, N_n$;

- We write the system N as a term $T[P_1, \ldots, P_n]$ to highlight the fact that system is based on the entities $P_1, \ldots, P_n$.

- Structural congruence rules defined on the forward system are in the following shape:

$$(\text{Struct}) \quad T[P_1, \ldots, P_n] \equiv T'[P_1, \ldots, P_n]$$

- The semantics of the system is given in terms of a reduction semantics with the following rule schema.

$$(\textsc{Act})\ P_1 \mid \ldots \mid P_n \twoheadrightarrow Q_1 \mid \ldots \mid Q_m$$

$$(\textsc{Opn})\ \frac{N_i \twoheadrightarrow N_i'}{opn(N_0, \ldots, N_i, \ldots, N_n) \twoheadrightarrow opn(N_0, \ldots, N_i', \ldots, N_n)}$$

$$(\textsc{Par})\ \frac{N_1 \twoheadrightarrow N_1'}{N_1 \mid N_2 \twoheadrightarrow N_1' \mid N_2} \qquad (\textsc{Equiv})\ \frac{N \equiv N' \quad N \twoheadrightarrow N_1 \quad N_1 \equiv N_1'}{N' \twoheadrightarrow N_1'}$$

DCore

Reversible semantics

- The reversible system R is defined as:

$$R ::= k : P \ \mid \ (R_1 \mid R_2) \ \mid opn(R_1, \dots, R_n) \mid H$$

$$H ::= [R_h; K]$$

where

- $k_1, \dots, k_n$ are unique keys identifying the entities;
- H is the memory called history, recording the past state of the system R_h and the set of the fresh keys generated with the forward execution, denoted with K.

- As in the forward model, we shall write the system R as a term

$T[k_1 : P_1, \dots, k_n : P_n]$ to highlight the fact that system is based on the identified entities $k_1 : P_1, \dots, k_n : P_n$.

- Structural congruence rules defined on the system are in the following shape:

$$\text{(Struct)} \quad T[k_1 : P_1, \dots, k_n : P_n] \equiv T'[k_1 : P_1, \dots, k_n : P_n]$$

The forward rules of the uncontrolled reversible semantics are:

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

$$(\text{F-Opn})\ \frac{R_i \twoheadrightarrow R_i' \mid H \quad (\texttt{key}(Ri') \setminus \texttt{key}(Ri)) \cap (\texttt{key}(R_0, \ldots, R_{i-1}, R_{i+1}, \ldots, R_n) = \emptyset}{opn(R_0, \ldots, R_i, \ldots, R_n) \twoheadrightarrow opn(R_0, \ldots, R_i', \ldots, R_n) \mid H}$$

$$(\text{F-Par})\ \frac{R_1 \twoheadrightarrow R_1' \quad (\texttt{key}(R_1') \setminus \texttt{key}(R_1)) \cap \texttt{key}(R_2) = \emptyset}{R_1 \mid R_2 \twoheadrightarrow R_1' \mid R_2} \qquad (\text{F-Equiv})\ \frac{R \equiv R' \quad R \twoheadrightarrow R_1 \quad R_1 \equiv R_1'}{R' \twoheadrightarrow R_1'}$$

- function $\texttt{key}(\cdot)$ compute the set of keys;
- in the rule (F-Act) the history is $H = [k_1 : P_1 \mid \ldots \mid k_n : P_n ; j_1, \ldots, j_m]$.

The backward rules of the uncontrolled reversible semantics are:

$$(\text{B-Act})\ \frac{H = [k_1 : P_1 \mid \ \ldots \ \mid k_n : P_n; j_1, \ldots, j_m]}{j_1 : Q_1 \mid \ \ldots \ \mid j_m : Q_m \mid H \rightsquigarrow k_1 : P_1 \mid \ \ldots \ \mid k_n : P_n}$$

$$(\text{B-Opn})\ \frac{R_i' \mid H \rightsquigarrow R_i}{opn(R_0, \ldots, R_i', \ldots, R_n) \mid H \rightsquigarrow opn(R_0, \ldots, R_i, \ldots, R_n)}$$

$$(\text{B-Par})\ \frac{R_1' \rightsquigarrow R_1}{R_1' \mid R_2 \rightsquigarrow R_1 \mid R_2} \qquad (\text{B-Equiv})\ \frac{R \equiv R' \quad R \rightsquigarrow R_1 \quad R_1 \equiv R_1'}{R' \rightsquigarrow R_1'}$$

Concurrent transitions

Definition

Given a history $H = [k_1 : P_1 \mid \ldots \mid k_n : P_n; K]$, the set of all keys belonging to the history H, written as $\texttt{key}(H)$, is defined with

$$\texttt{key}(H) = \{k_1\} \cup \cdots \cup \{k_n\} \cup K.$$

Transitions are labeled with H and we have that transition t of a system R is $t : R \xrightarrow{H} R'$, where H is the memory created or consumed by transition.

Definition (Concurrent transitions)

Two coinitial transitions $R \xrightarrow{H'} R'$ and $R \xrightarrow{H''} R''$ are *concurrent* if $\texttt{key}(H') \cap \texttt{key}(H'') = \emptyset$. Transitions which are not concurrent are in *conflict*.

DCore

Properties

Lemma (Loop Lemma)

For every reachable system R and forward transition $t : R \overset{H}{\twoheadrightarrow} R'$, there exists the backward transition $t^\bullet : R' \overset{H}{\rightsquigarrow} R$ and vice versa.

Lemma (Square Lemma)

If $t_1 : R \xrightarrow{H'} R'$ and $t_2 : R \xrightarrow{H''} R''$ are two coinitial concurrent transitions, there exist two cofinal transitions $t_2/t_1 : R' \xrightarrow{H''} R'''$ and $t_1/t_2 : R'' \xrightarrow{H'} R'''$.

Theorem (Causal Consistency)

Given two coinitial derivations d_1 and d_2, $d_1 \sim d_2$ if and only if d_1 and d_2 are cofinal.

- Danos, V., Krivine, J.: Reversible communicating systems. In: CONCUR. LNCS, vol. 3170, pp. 292–307. Springer (2004)

DCore

Case Studies: Higher-Order π-calculus

Example:

- **Higher-Order π-calculus**

$a\langle P_1 \mid P_2\rangle \mid a(X) \triangleright X \to P_1 \mid P_2$

- **$\rho\pi$ calculus**

$$\begin{aligned}(k_1 : a\langle P_1 \mid P_2\rangle) \mid (k_2 : a(X) \triangleright X) &\to \nu k.k : (P_1 \mid P_2) \mid [M; k] \\ &\equiv \nu k, h_1, h_2.(\langle h_1, \tilde{h}\rangle \cdot k : P_1 \mid \langle h_2, \tilde{h}\rangle \cdot k : P_2)\end{aligned}$$

where $M = (k_1 : a\langle P_1 \mid P_2\rangle) \mid (k_2 : a(X) \triangleright X)$ and $\tilde{h} = \{h_1, h_2\}$

- **Our reversible semantics for HOπ**

$k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X \twoheadrightarrow j_1 : P_1 \mid j_2 : P_2 \mid [R_h; \{j_1, j_2\}]$

where $R_h = k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X$

Case Studies: Erlang

The forward Erlang system is defined as a pool of processes and floating messages with the following grammar:

$$N := \langle p, \theta, e \rangle \mid (p, p', v) \mid \; (N_1 \mid N_2)$$

where

- $\langle p, \theta, e \rangle$ represents a process with its pid p, environment θ and an expression e to be evaluated;
- every process has a unique pid;
- (p, p', v) represents a floating message with content v sent from the process with pid p to the one with pid p'.
  Floating messages are the messages in the system after they are sent and before they are received.

DCore

Erlang semantics:

$$(\textsc{Seq})\ \frac{\theta, e \xrightarrow{\tau} \theta', e'}{\langle p, \theta, e\rangle \hookrightarrow \langle p, \theta', e'\rangle}$$

$$(\textsc{Rec})\ \frac{\theta, e \xrightarrow{\texttt{rec}(\kappa,\overline{cl_n})} \theta', e' \quad \text{and} \quad \texttt{matchrec}(\theta, \overline{cl_n}, v) = (\theta_i, e_i)}{(p', p, \{v, k''\}) \mid \langle p, \theta, e\rangle \hookrightarrow \langle p, \theta'\theta_i, e'\{\kappa \mapsto e_i\}\rangle}$$

$$(\textsc{Send})\ \frac{\theta, e \xrightarrow{\texttt{send}(p',v)} \theta', e' \quad k'' \text{ is a fresh symbol}}{\langle p, \theta, e\rangle \hookrightarrow \langle p, \theta', e'\rangle \mid (p, p', \{v, k''\})}$$

$$(\textsc{Spawn})\ \frac{\theta, e \xrightarrow{\texttt{spawn}(\kappa, f/n, [\overline{v_n}])} \theta', e' \quad p' \text{ is a fresh pid}}{\langle p, \theta, e\rangle \hookrightarrow \langle p, \theta', e'\{\kappa \mapsto p'\}\rangle \mid \langle p', id, \texttt{apply}\ f/n\ (\overline{v_n})\rangle}$$

$$(\textsc{Self})\ \frac{\theta, e \xrightarrow{\texttt{self}(\kappa)} \theta', e'}{\langle p, \theta, e\rangle \hookrightarrow \langle p, \theta', e'\{\kappa \mapsto p\}\rangle} \qquad (\textsc{Par})\ \frac{N \hookrightarrow N' \quad \texttt{pid}(N') \cap \texttt{pid}(N_1) = \emptyset}{N \mid N_1 \hookrightarrow N' \mid N_1}$$

- Ivan Lanese, Adrián Palacios and Germán Vidal (2019): Causal-consistent replay debugging for message passing programs. In: Technical report, DSIC, Universitat Politecnica de Valencia

Reversible Erlang semantics

The reversible Erlang system is defined as:

$$R := k : \langle p, \theta, e \rangle \mid k : (p, p', v) \mid \ (R \mid H) \ \mid \ (R_1 \mid R_2)$$

where

- k is the unique key identifying the entity;
- H is a history

DCore

Uncontrolled reversible forward semantics:

$$(\text{F-Seq})\ \frac{\theta, e \xrightarrow{\tau} \theta', e' \quad k' \text{ is a fresh key}}{k : \langle p, \theta, e\rangle \twoheadrightarrow k' : \langle p, \theta', e'\rangle \mid [k : \langle p, \theta, e\rangle; k']}$$

$$(\text{F-Send})\ \frac{\theta, e \xrightarrow{\mathtt{send}(p',v)} \theta', e' \quad k', k'' \text{ are fresh keys}}{k : \langle p, \theta, e\rangle \twoheadrightarrow k' : \langle p, \theta', e'\rangle \mid k'' : (p, p', v) \mid [k : \langle p, \theta, e\rangle; k', k'']}$$

$$(\text{F-Rec})\ \frac{\theta, e \xrightarrow{\mathtt{rec}(\kappa,\overline{cl_n})} \theta', e' \quad \text{and} \quad \mathtt{matchrec}(\theta, \overline{cl_n}, v) = (\theta_i, e_i) \quad k' \text{ is a fresh key}}{k'' : (p', p, v) \mid k : \langle p, \theta, e\rangle \twoheadrightarrow k' : \langle p, \theta'\theta_i, e'\{\kappa \mapsto e_i\}\rangle \mid [k'' : (p', p, v) \mid k : \langle p, \theta, e\rangle; k']}$$

$$(\text{F-Spawn})\ \frac{\theta, e \xrightarrow{\mathtt{spawn}(\kappa, f/n, [\overline{v_n}])} \theta', e' \quad p' \text{ is a fresh pid,} \quad \text{k',k'' are fresh keys}}{k : \langle p, \theta, e\rangle \twoheadrightarrow k' : \langle p, \theta', e'\{\kappa \mapsto p'\}\rangle \mid k'' : \langle p', id, \text{apply } f/n\ (\overline{v_n})\rangle \mid [k : \langle p, \theta, e\rangle; k', k'']}$$

$$(\text{F-Self})\ \frac{\theta, e \xrightarrow{\mathtt{self}(\kappa)} \theta', e' \quad k' \text{ is a fresh key}}{k : \langle p, \theta, e\rangle \twoheadrightarrow k' : \langle p, \theta', e'\{\kappa \mapsto p\}\rangle \mid [k : \langle p, \theta, e\rangle; k']}$$

$$(\text{F-Par})\ \frac{R \twoheadrightarrow R' \mid H \quad \mathtt{pid}(R') \cap \mathtt{pid}(R_1) = \emptyset \quad \text{and} \quad (\mathtt{key}(R') \setminus \mathtt{key}(R)) \cap \mathtt{key}(R_1) = \emptyset}{R \mid R_1 \twoheadrightarrow R' \mid R_1 \mid H}$$

Correspondence between two reversible semantics for Erlang:

- we show that if two transitions are concurrent in the uncontrolled reversible logging semantics [1], they are concurrent in our reversible semantics (i.e. we prove the correspondence between two conflict relations):

Theorem

Given two different coinitial transitions t_1 and t_2. They are in conflict in the uncontrolled reversible logging semantics [1] *if and only if they are in conflict in our uncontrolled reversible semantics.*

- we proved that two reversible semantics are strong backward and forward barbed equivalent.

Theorem

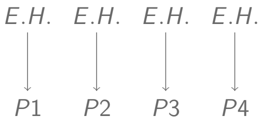
The reversible logging semantics of [1] *and our reversible semantics are strong backward and forward barbed equivalent.*

[1] Ivan Lanese, Adrián Palacios and Germán Vidal (2019): Causal-consistent replay debugging for message passing programs. In: Technical report, DSIC, Universitat Politecnica de Valencia

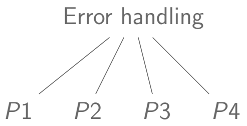
DCore

Reversible Link Semantics for Erlang (work in progress)

- What is a link?
  - Link can be seen as a bidirectional path between two processes along which error signals travel.
- Why links?
  - Links are one of the main factors standing behind Erlang's remote error handling mechanisms.
  - Remote error handling is an important feature of Erlang. In contrast to Local error handling, the systems have a higher degree of separation of concerns.



Local Error Handling



Remote Error Handling

Reversible Link Semantics for Erlang: (work in progress)

Differences with the simple Erlang model (given few slides before):

- the syntax of Erlang is enriched with the function `spawn_link`();
- the process needs to keep track of all the processes linked with it i.e. process becomes defined as $\langle p, \theta, e, l \rangle$, where l is a set of links;
- notion of *terminated process*.
  The process $\langle p, \theta, e, l \rangle$ terminated normally if $e = v$ or it terminated abnormally if $e = r$.
- the forward system is defined as:

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

- the reversible Link system for Erlang is defined as:

$$R := k : \langle p, \theta, e, l \rangle \mid k : (p, p', v) \mid \ (R \mid H) \ \mid \ (R_1 \mid R_2)$$

DCore

New rules for the link semantics:

- High-level semantics (system rules):

$$(\text{F-ErrP})\ \frac{e \in \{v, r\} \quad \wedge \quad p_1 \in l \quad \wedge \quad k', k_1', k_2 \text{ are fresh keys}}{\begin{array}{c} k : \langle p, \theta, e, l \rangle \mid k_1 : \langle p_1, \theta_1, e_1, l_1 \rangle \twoheadrightarrow k' : \langle p, \theta, e, l \setminus p_1 \rangle \mid k_1' : \langle p_1, \theta_1, e_1, l_1 \setminus p \rangle \mid k_2 : (p, p_1, e) \mid \\ [k : \langle p, \theta, e, l \rangle \mid k_1 : \langle p_1, \theta_1, e_1, l_1 \rangle; k', k_1', k_2] \end{array}}$$

$$(\text{F-SpLink})\ \frac{\theta, e \xrightarrow{\texttt{spawn\_link}(\kappa, f/n, [\overline{v_n}])} \theta', e' \qquad p' \text{ is a fresh pid} \qquad k', k'' \text{ are fresh keys}}{\begin{array}{c} k : \langle p, \theta, e, l \rangle \twoheadrightarrow k' : \langle p, \theta', e'\{\kappa \mapsto p'\}, l \cup \{\kappa \mapsto p'\} \rangle \mid k'' : \langle p', id, \texttt{apply } f/n\ (\overline{v_n}), \emptyset \cup p \rangle \mid \\ [k : \langle p, \theta, e, l \rangle; k', k''] \end{array}}$$

New rules for the link semantics:

- The semantics of evaluation of sequential expressions [1]:

$$(\textsc{Call3})\ \frac{\mathtt{eval}(op, v_1, \ldots, v_n) = r}{\theta, \mathtt{call}\ op(v_1, \ldots, v_n) \xrightarrow{\tau} \theta, r}$$

-In this way we capture the error in built in functions.

- The semantics concerning the evaluation of the concurrent expressions of [1]:

-the new rules for the function `spawn_link()` are very similar to the rules for function `spawn()`.

[1] Ivan Lanese, Adrián Palacios and Germán Vidal (2019): Causal-consistent replay debugging for message passing programs. In: Technical report, DSIC, Universitat Politecnica de Valencia

Current and future work:

Current work:

- We are working on defining the Reversible Link Semantics for Erlang;
- We are finalising proofs;

Future work:

- Investigating if our approach (with some modifications) could be applied to some other models as Klaim;
- Adding the link semantics into CauDEr.