Towards Reversible Computation in Erlang

Adrián Palacios
(joint work with Naoki Nishida and Germán Vidal)

Technical University of Valencia

26th International Symposium on
Logic-Based Program Synthesis and Transformation

September 8, 2016

Edinburgh, United Kingdom

Reversible Computation

A computation principle is reversible if any forward computation can be undone by a finite sequence of backward steps

Applications

- debugging
- enforcing fault-tolerance
- quantum computing
- ...

Landauer's embedding

An irreversible computation can be turned into a reversible one if we store the *history* of the computation.

Reversible Computation

A computation principle is reversible if any forward computation can be undone by a finite sequence of backward steps

Applications

- debugging
- enforcing fault-tolerance
- quantum computing
- ...

Landauer's embedding

An irreversible computation can be turned into a reversible one if we store the *history* of the computation.

Motivation

We have recently introduced a reversible term rewriting principle.

We want to consider a real programming language: **Erlang**

Why Erlang?

- Introduces concurrency, a challenge for reversibility
- Used in large-scale distributed systems
- Reversible Computation can contribute to its reliability (debugging, safe sessions, etc.)

Motivation

We have recently introduced a reversible term rewriting principle.

We want to consider a real programming language: **Erlang**

Why Erlang?

- Introduces concurrency, a challenge for reversibility
- Used in large-scale distributed systems
- Reversible Computation can contribute to its reliability (debugging, safe sessions, etc.)

Erlang

Erlang's features

Main features of Erlang:

- integration of functional and concurrent features
- concurrency model based on asynchronous message-passing
- dynamic typing
- hot code loading

These features make it appropiate for distributed, fault-tolerant applications (Facebook, WhatsApp)

Erlang syntax

We consider a subset of Erlang with this syntax:

$$
\begin{array}{rcl}
Module & ::= & \text{module } Atom = fun_1, \ldots, fun_n \\
fun & ::= & fname = \text{fun } (X_1, \ldots, X_n) \to expr \\
fname & ::= & Atom/Integer \\
lit & ::= & Atom \mid Integer \mid Float \mid [\,] \\
expr & ::= & Var \mid lit \mid fname \mid [expr_1|expr_2] \mid \{expr_1, \ldots, expr_n\} \\
& \mid & \text{call } expr\ (expr_1, \ldots, expr_n) \mid \text{apply } expr\ (expr_1, \ldots, expr_n) \\
& \mid & \text{case } expr \text{ of } clause_1; \ldots; clause_m \text{ end} \\
& \mid & \text{let } Var = expr_1 \text{ in } expr_2 \mid \text{receive } clause_1; \ldots; clause_n \text{ end} \\
& \mid & \text{spawn}(expr, [expr_1, \ldots, expr_n]) \mid expr_1 \,!\, expr_2 \mid \text{self}() \\
clause & ::= & pat \text{ when } expr_1 \to expr_2 \\
pat & ::= & Var \mid lit \mid [pat_1|pat_2] \mid \{pat_1, \ldots, pat_n\}
\end{array}
$$

Erlang syntax

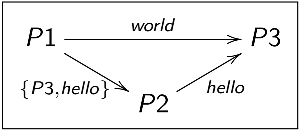
We consider a subset of Erlang with this syntax:

$$
\begin{array}{rcl}
Module & ::= & \text{module } Atom = fun_1, \ldots, fun_n \\
fun & ::= & fname = \text{fun } (X_1, \ldots, X_n) \to expr \\
fname & ::= & Atom/Integer \\
lit & ::= & Atom \mid Integer \mid Float \mid [\,] \\
expr & ::= & Var \mid lit \mid fname \mid [expr_1|expr_2] \mid \{expr_1, \ldots, expr_n\} \\
 & \mid & \text{call } expr\ (expr_1, \ldots, expr_n) \mid \text{apply } expr\ (expr_1, \ldots, expr_n) \\
 & \mid & \text{case } expr \text{ of } clause_1; \ldots; clause_m \text{ end} \\
 & \mid & \text{let } Var = expr_1 \text{ in } expr_2 \mid \text{receive } clause_1; \ldots; clause_n \text{ end} \\
 & \mid & \text{spawn}(expr, [expr_1, \ldots, expr_n]) \mid expr_1 \,!\, expr_2 \mid \text{self}() \\
clause & ::= & pat \text{ when } expr_1 \to expr_2 \\
pat & ::= & Var \mid lit \mid [pat_1|pat_2] \mid \{pat_1, \ldots, pat_n\}
\end{array}
$$

Example

$$
\begin{aligned}
main/0 = \text{fun } () \rightarrow\ & \text{let } P2 = \text{spawn}(echo/0, [\,]) \\
& \text{in let } P3 = \text{spawn}(target/0, [\,]) \\
& \text{in let } \_ = P3 \,!\, world \\
& \text{in let } P2 \,!\, \{P3, hello\}
\end{aligned}
$$

$$
\begin{aligned}
target/0 = \text{fun } () \rightarrow\ & \text{receive} \\
& \quad A \rightarrow \text{receive} \\
& \quad\quad\quad B \rightarrow \{A, B\} \\
& \quad\quad \text{end} \\
& \text{end}
\end{aligned}
$$

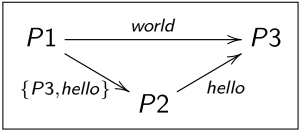


$$
\begin{aligned}
echo/0 = \text{fun } () \rightarrow\ & \text{receive} \\
& \quad \{P, M\} \rightarrow P \,!\, M \\
& \text{end}
\end{aligned}
$$

Example

$$
\begin{array}{l}
main/0 = \mathrm{fun}\ () \rightarrow \mathrm{let}\ P2 = \mathrm{spawn}(echo/0, [\,])\\
\qquad\qquad\qquad\quad \mathrm{in\ let}\ P3 = \mathrm{spawn}(target/0, [\,])\\
\qquad\qquad\qquad\quad \mathrm{in\ let}\ \_ = P3\,!\,world\\
\qquad\qquad\qquad\quad \mathrm{in\ let}\ P2\,!\,\{P3, hello\}
\end{array}
$$

$$
\begin{array}{l}
target/0 = \mathrm{fun}\ () \rightarrow \mathrm{receive}\\
\qquad\qquad\qquad\qquad A \rightarrow \mathrm{receive}\\
\qquad\qquad\qquad\qquad\qquad\quad B \rightarrow \{A, B\}\\
\qquad\qquad\qquad\qquad\qquad\ \mathrm{end}\\
\qquad\qquad\qquad\quad\ \mathrm{end}
\end{array}
$$

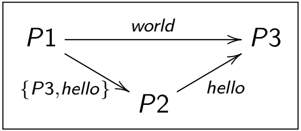


$$
\begin{array}{l}
echo/0 = \mathrm{fun}\ () \rightarrow \mathrm{receive}\\
\qquad\qquad\qquad\qquad \{P, M\} \rightarrow P\,!\,M\\
\qquad\qquad\qquad\quad\ \mathrm{end}
\end{array}
$$

Example

$$
\begin{aligned}
main/0 = \text{fun } () \rightarrow\ & \text{let } P2 = \text{spawn}(echo/0, [\,]) \\
& \text{in let } P3 = \text{spawn}(target/0, [\,]) \\
& \text{in let } \_ = {\color{red} P3 \,!\, world} \\
& \text{in let } P2 \,!\, \{P3, hello\}
\end{aligned}
$$

$$
\begin{aligned}
target/0 = \text{fun } () \rightarrow\ & \text{receive} \\
& \quad A \rightarrow \text{receive} \\
& \qquad\qquad B \rightarrow \{A, B\} \\
& \qquad\quad\ \text{end} \\
& \text{end}
\end{aligned}
$$

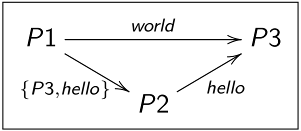


$$
\begin{aligned}
echo/0 = \text{fun } () \rightarrow\ & \text{receive} \\
& \quad \{P, M\} \rightarrow P \,!\, M \\
& \text{end}
\end{aligned}
$$

Example

$$
\begin{array}{l}
main/0 = \text{fun } () \rightarrow \text{let } P2 = \text{spawn}(echo/0, [\,]) \\
\qquad\qquad\qquad\qquad \text{in let } P3 = \text{spawn}(target/0, [\,]) \\
\qquad\qquad\qquad\qquad \text{in let } \_ = P3 \,!\, world \\
\qquad\qquad\qquad\qquad \text{in let } P2 \,!\, \{P3, hello\}
\end{array}
$$

$$
\begin{array}{l}
target/0 = \text{fun } () \rightarrow \text{receive} \\
\qquad\qquad\qquad\qquad A \rightarrow \text{receive} \\
\qquad\qquad\qquad\qquad\qquad B \rightarrow \{A, B\} \\
\qquad\qquad\qquad\qquad\quad \text{end} \\
\qquad\qquad\qquad\quad \text{end}
\end{array}
$$



$$
\begin{array}{l}
echo/0 = \text{fun } () \rightarrow \text{receive} \\
\qquad\qquad\qquad\qquad \{P, M\} \rightarrow P \,!\, M \\
\qquad\qquad\qquad\quad \text{end}
\end{array}
$$

Semantics

Previous definitions

Definition (process)

A process is a triple $\langle \mathrm{p}, (\theta, e), q \rangle$ where

- p is the pid of the process
- (θ, e) is the control of the state
- q is the process' mailbox

Definition (system)

A system is denoted by $\Gamma; \Pi$, where

- Π is a pool of processes
- Γ is the global mailbox of the system

We often use $\Gamma; \langle \mathrm{p}, (\theta, e), q \rangle \& \Pi$

Semantics of expressions: sequential actions

$$(\mathit{Var})\ \frac{}{\theta, X \xrightarrow{\tau} \theta, \theta(X)} \qquad (\mathit{Tuple})\ \frac{\theta, e_i \xrightarrow{\ell} \theta', e'_i \quad i \in \{1, \ldots, n\}}{\theta, \{e_1, \ldots, e_n\} \xrightarrow{\ell} \theta', \{\overline{e_{1,i-1}}, e'_i, \overline{e_{i+1,n}}\}}$$

$$(\mathit{List1})\ \frac{\theta, e_1 \xrightarrow{\ell} \theta', e'_1}{\theta, [e_1|e_2] \xrightarrow{\ell} \theta', [e'_1|e_2]} \qquad (\mathit{List2})\ \frac{\theta, e_2 \xrightarrow{\ell} \theta', e'_2}{\theta, [e_1|e_2] \xrightarrow{\ell} \theta', [e_1|e'_2]}$$

$$(\mathit{Let1})\ \frac{\theta, e_1 \xrightarrow{\ell} \theta', e'_1}{\theta, \mathsf{let}\ X = e_1\ \mathsf{in}\ e_2 \xrightarrow{\ell} \theta', \mathsf{let}\ X = e'_1\ \mathsf{in}\ e_2} \qquad (\mathit{Let2})\ \frac{}{\theta, \mathsf{let}\ X = v\ \mathsf{in}\ e \xrightarrow{\tau} \theta[X \mapsto v], e}$$

$$(\mathit{Case1})\ \frac{\theta, e \xrightarrow{\ell} \theta', e'}{\begin{array}{l}\theta, \mathsf{case}\ e\ \mathsf{of}\ cl_1; \ldots; cl_n\ \mathsf{end} \\ \xrightarrow{\ell} \theta', \mathsf{case}\ e'\ \mathsf{of}\ cl_1; \ldots; cl_n\ \mathsf{end}\end{array}} \qquad (\mathit{Case2})\ \frac{\mathsf{match}(v, cl_1, \ldots, cl_n) = \langle \theta_i, e_i \rangle}{\theta, \mathsf{case}\ v\ \mathsf{of}\ cl_1; \ldots; cl_n\ \mathsf{end} \xrightarrow{\tau} \theta\theta_i, e_i}$$

$$(\mathit{Apply1})\ \frac{\theta, e_i \xrightarrow{\ell} \theta', e'_i \quad i \in \{1, \ldots, n\}}{\theta, \mathsf{apply}\ a/n\ (\overline{e_n}) \xrightarrow{\ell} \theta', \mathsf{apply}\ a/n\ (\overline{e_{1,i-1}}, e'_i, \overline{e_{i+1,n}})}$$

$$(\mathit{Apply2})\ \frac{\mu(a/n) = \mathsf{fun}\ (X_1, \ldots, X_n) \to e}{\theta, \mathsf{apply}\ a/n\ (v_1, \ldots, v_n) \xrightarrow{\tau} \{X_1 \mapsto v_1, \ldots, X_n \mapsto v_n\}, e}$$

Semantics of expressions: concurrent actions

$$(\mathit{Send1})\ \frac{\theta, e_1 \xrightarrow{\ell} \theta', e_1'}{\theta, e_1\ !\ e_2 \xrightarrow{\ell} \theta', e_1'\ !\ e_2} \qquad \frac{\theta, e_2 \xrightarrow{\ell} \theta', e_2'}{\theta, e_1\ !\ e_2 \xrightarrow{\ell} \theta', e_1\ !\ e_2'}$$

$$(\mathit{Send2})\ \frac{}{\theta, v_1\ !\ v_2 \xrightarrow{\mathsf{send}(v_1,v_2)} \theta, v_2}$$

$$(\mathit{Receive})\ \frac{}{\theta, \mathsf{receive}\ cl_1; \ldots; cl_n\ \mathsf{end} \xrightarrow{\mathsf{rec}(y,\overline{cl_n})} \theta, y}$$

$$(\mathit{Spawn})\ \frac{}{\theta, \mathsf{spawn}(a/n, [e_1, \ldots, e_n]) \xrightarrow{\mathsf{spawn}(y,a/n,[\overline{e_n}])} \theta, y}$$

$$(\mathit{Self})\ \frac{}{\theta, \mathsf{self}() \xrightarrow{\mathsf{self}(y)} \theta, y}$$

System semantics

(*Exp*) $$\frac{\theta, e \xrightarrow{\tau} \theta', e'}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma; \langle \mathrm{p}, (\theta', e'), q\rangle \& \Pi}$$

(*Send*) $$\frac{\theta, e \xrightarrow{\mathsf{send}(\mathrm{p}'', v)} \theta', e'}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma \cup (\mathrm{p}, \mathrm{p}'', v); \langle \mathrm{p}, (\theta', e'), q\rangle \& \Pi}$$

(*Receive*) $$\frac{\theta, e \xrightarrow{\mathsf{rec}(y, \overline{cl_n})} \theta', e' \quad \mathsf{matchrec}(\overline{cl_n}, q) = (\theta_i, e_i, q')}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma; \langle \mathrm{p}, (\theta'\theta_i, e'\{y \mapsto e_i\}), q'\rangle \& \Pi}$$

(*Spawn*) $$\frac{\theta, e \xrightarrow{\mathsf{spawn}(y, a/n, [\overline{e_n}])} \theta', e' \quad \mathrm{p}' \text{ is a fresh pid}}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma; \langle \mathrm{p}, (\theta', e'\{y \mapsto \mathrm{p}'\}), q\rangle \& \langle \mathrm{p}', (\theta', \mathsf{apply}\ a/n\ (e_1, \ldots, e_n)), [\,]\rangle}$$

(*Self*) $$\frac{\theta, e \xrightarrow{\mathsf{self}(y)} \theta', e'}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma; \langle \mathrm{p}, (\theta', e'\{y \mapsto \mathrm{p}\}), q\rangle \& \Pi}$$

(*Sched*) $$\frac{\alpha(\Gamma) = (\mathrm{p}', \mathrm{p}) \quad \Pi = \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi'}{\Gamma; \Pi \longmapsto \Gamma \setminus \{(\mathrm{p}', \mathrm{p}, v)\}; \langle \mathrm{p}, (\theta, e), v : q\rangle \& \Pi'}$$

System semantics

(*Exp*)
$$\frac{\theta, e \xrightarrow{\tau} \theta', e'}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma; \langle \mathrm{p}, (\theta', e'), q\rangle \& \Pi}$$

(*Send*)
$$\frac{\theta, e \xrightarrow{\mathsf{send}(\mathrm{p}'', v)} \theta', e'}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma \cup (\mathrm{p}, \mathrm{p}'', v); \langle \mathrm{p}, (\theta', e'), q\rangle \& \Pi}$$

(*Receive*)
$$\frac{\theta, e \xrightarrow{\mathsf{rec}(y, \overline{cl_n})} \theta', e' \quad \mathsf{matchrec}(\overline{cl_n}, q) = (\theta_i, e_i, q')}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma; \langle \mathrm{p}, (\theta'\theta_i, e'\{y \mapsto e_i\}), q'\rangle \& \Pi}$$

(*Spawn*)
$$\frac{\theta, e \xrightarrow{\mathsf{spawn}(y, a/n, [\overline{e_n}])} \theta', e' \quad \mathrm{p}' \text{ is a fresh pid}}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma; \langle \mathrm{p}, (\theta', e'\{y \mapsto \mathrm{p}'\}), q\rangle \& \langle \mathrm{p}', (\theta', \mathsf{apply}\ a/n\ (e_1, \ldots, e_n)), [\,]\rangle}$$

(*Self*)
$$\frac{\theta, e \xrightarrow{\mathsf{self}(y)} \theta', e'}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma; \langle \mathrm{p}, (\theta', e'\{y \mapsto \mathrm{p}\}), q\rangle \& \Pi}$$

(*Sched*)
$$\frac{\alpha(\Gamma) = (\mathrm{p}', \mathrm{p}) \qquad \Pi = \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi'}{\Gamma; \Pi \longmapsto \Gamma \setminus \{(\mathrm{p}', \mathrm{p}, v)\}; \langle \mathrm{p}, (\theta, e), v : q\rangle \& \Pi'}$$

System semantics

$$(\mathit{Exp})\quad \frac{\theta, e \xrightarrow{\tau} \theta', e'}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma; \langle \mathrm{p}, (\theta', e'), q\rangle \& \Pi}$$

$$(\mathit{Send})\quad \frac{\theta, e \xrightarrow{\mathsf{send}(\mathrm{p}'', v)} \theta', e'}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma \cup (\mathrm{p}, \mathrm{p}'', v); \langle \mathrm{p}, (\theta', e'), q\rangle \& \Pi}$$

$$(\mathit{Receive})\quad \frac{\theta, e \xrightarrow{\mathsf{rec}(y, \overline{cl_n})} \theta', e' \quad \mathsf{matchrec}(\overline{cl_n}, q) = (\theta_i, e_i, q')}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma; \langle \mathrm{p}, (\theta'\theta_i, e'\{y \mapsto e_i\}), q'\rangle \& \Pi}$$

$$(\mathit{Spawn})\quad \frac{\theta, e \xrightarrow{\mathsf{spawn}(y, a/n, [\overline{e_n}])} \theta', e' \quad \mathrm{p}' \text{ is a fresh pid}}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma; \langle \mathrm{p}, (\theta', e'\{y \mapsto \mathrm{p}'\}), q\rangle \& \langle \mathrm{p}', (\theta', \mathsf{apply}\ a/n\ (e_1, \ldots, e_n)), [\,]\rangle}$$

$$(\mathit{Self})\quad \frac{\theta, e \xrightarrow{\mathsf{self}(y)} \theta', e'}{\Gamma; \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi \longmapsto \Gamma; \langle \mathrm{p}, (\theta', e'\{y \mapsto \mathrm{p}\}), q\rangle \& \Pi}$$

$$(\mathit{Sched})\quad \frac{\alpha(\Gamma) = (\mathrm{p}', \mathrm{p}) \qquad \Pi = \langle \mathrm{p}, (\theta, e), q\rangle \& \Pi'}{\Gamma; \Pi \longmapsto \Gamma \setminus \{(\mathrm{p}', \mathrm{p}, v)\}; \langle \mathrm{p}, (\theta, e), v : q\rangle \& \Pi'}$$

Reversible Semantics

Landauer's embedding

Checkpoints

We use **checkpoints** to stop backward computation: $\#_k^{\mathsf{t}}$

- k is a label (identifying the kind of checkpoint)
- t is a unique identifier

there are **three kinds** of checkpoins:

- introduced by the programmer: $\#_{\mathsf{ch}}$

$$\text{let } \_ = \text{check in } expr$$

- internal: $\#_\alpha$ (message dispatching) and $\#_{\mathsf{sp}}$ (spawning a process)

Forward (reversible) semantics

$$(\mathit{Internal})\quad \frac{\theta, e \xrightarrow{\tau} \theta', e'}{\Gamma; \langle \pi, \mathrm{p}, (\theta, e), q\rangle \& \Pi \rightharpoonup \Gamma; \langle \tau(\theta, e) : \pi, \mathrm{p}, (\theta', e'), q\rangle \& \Pi}$$

$$(\mathit{Check})\quad \frac{\theta, e \xrightarrow{\mathsf{check}(y)} \theta', e' \quad \text{and } \mathtt{t} \text{ is fresh}}{\Gamma; \langle \pi, \mathrm{p}, (\theta, e), q\rangle \& \Pi \rightharpoonup \Gamma; \langle \mathsf{check}(\theta, e) : \#^{\mathtt{t}}_{\mathsf{ch}} : \pi, \mathrm{p}, (\theta', e'\{y \mapsto \mathtt{t}\}), q\rangle \& \Pi}$$

$$(\mathit{Self})\quad \frac{\theta, e \xrightarrow{\mathsf{self}(y)} \theta', e'}{\Gamma; \langle \pi, \mathrm{p}, (\theta, e), q\rangle \& \Pi \rightharpoonup \Gamma; \langle \mathsf{self}(\theta, e) : \pi, \mathrm{p}, (\theta', e'\{y \mapsto \mathrm{p}\}), q\rangle \& \Pi}$$

Forward (reversible) semantics (cont.)

$$(\mathit{Spawn})\quad \frac{\theta, e \xrightarrow{\mathsf{spawn}(y,a/n,[e_1,\ldots,e_n])} \theta', e' \quad \mathrm{p}' \text{ is a fresh pid} \quad \text{and } \mathtt{t} \text{ is fresh}}{\begin{array}{ll}\Gamma; \langle \pi, \mathrm{p}, (\theta, e), q\rangle \& \Pi \rightharpoonup & \Gamma; \langle \mathsf{spawn}(\theta, e, \mathrm{p}') : \pi, \mathrm{p}, (\theta', e'\{y \mapsto \mathrm{p}'\}), q\rangle \\ & \& \langle [\,], \mathrm{p}', (\theta, (\mathsf{apply}\ a/n\ (e_1, \ldots, e_n)), [\,]\rangle \& \Pi\end{array}}$$

$$(\mathit{Send})\quad \frac{\theta, e \xrightarrow{\mathsf{send}(\mathrm{p}'', v)} \theta', e' \quad \text{and } \mathtt{t} \text{ is fresh}}{\Gamma; \langle \pi, \mathrm{p}, (\theta, e), q\rangle \& \Pi \rightharpoonup \Gamma \cup (\mathrm{p}, \mathrm{p}'', \{\mathtt{t}, v\}); \langle \mathsf{send}(\mathrm{p}'', \theta, e, \mathtt{t}) : \pi, \mathrm{p}, (\theta', e'), q\rangle \& \Pi}$$

$$(\mathit{Receive})\quad \frac{\theta, e \xrightarrow{\mathsf{rec}(y, \overline{cl_n})} \theta', e' \quad \mathsf{matchrec}(\overline{cl_n}, q) = (\theta_i, e_i, q', \mathtt{t})}{\Gamma; \langle \pi, \mathrm{p}, (\theta, e), q\rangle \& \Pi \rightharpoonup \Gamma; \langle \mathsf{rec}(\theta, e, q) : \pi, \mathrm{p}, (\theta'\theta_i, e'\{y \mapsto e_i\}), q'\rangle \& \Pi}$$

$$(\mathit{Sched})\quad \frac{\alpha(\Gamma) = (\mathrm{p}', \mathrm{p}) \quad \Pi = \langle \pi, \mathrm{p}, (\theta, e), q\rangle \& \Pi'}{\Gamma; \Pi \rightharpoonup \Gamma \setminus (\mathrm{p}', \mathrm{p}, \{\mathtt{t}, v\}); \langle \alpha(\mathrm{p}', \mathrm{p}, \{\mathtt{t}, v\}) : \pi, \mathrm{p}, (\theta, e), \{\mathtt{t}, v\} : q\rangle \& \Pi'}$$

Backward semantics

Causal consistency

An action cannot be undone until all the actions that depend on it have been already undone

E.g., if a process spawns another process, we cannot undo this process spawning until all the actions performed by the new process are undone

Backward semantics (1)

We follow a rollback fashion:

$$(\mathit{Undo1}) \quad \Gamma; \langle \pi, \mathrm{p}, (\theta, e), q \rangle \And \Pi \leftharpoondown \Gamma; \lfloor \langle \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\#^{\mathtt{t}}_{\mathsf{ch}}} \And \Pi$$

$$(\mathit{Undo2}) \quad \Gamma; \lfloor \langle \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\Psi} \And \Pi \leftharpoondown \Gamma; \lfloor \langle \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\Psi \cup \#^{\mathtt{t}}_{\mathsf{ch}}} \And \Pi$$

where Ψ is the set of *pending* checkpoints

Backward semantics (2)

Then, we apply the backward semantics (system rules)

- Sending a message: propagates the rollback to the receiver
- Spawning a process: propagates the rollback to the new process

$$
\begin{array}{l}
(\textit{Send2}) \\
\quad \Gamma; \lfloor\langle \mathsf{send}(\mathrm{p}'', \theta, \mathsf{e}, \mathtt{t}) : \pi, \mathrm{p}, (\theta', \mathsf{e}'), q\rangle\rfloor_{\Psi} \;\&\; \langle \pi'', \mathrm{p}'', (\theta'', \mathsf{e}''), q''\rangle \;\&\; \Pi \\
\qquad\qquad\qquad\qquad \leftharpoondown \Gamma; \lfloor\langle \pi, \mathrm{p}, (\theta, \mathsf{e}), q\rangle\rfloor_{\Psi} \;\&\; \lfloor\langle \pi'', \mathrm{p}'', (\theta'', \mathsf{e}''), q''\rangle\rfloor_{\#^{\mathtt{t}}_{\alpha}} \;\&\; \Pi \\
\qquad\qquad\qquad\qquad\qquad \text{if } (\mathrm{p}, \mathrm{p}'', \{\mathtt{t}, v\}) \text{ does not occur in } \Gamma
\end{array}
$$

$$
\begin{array}{l}
(\textit{Spawn}) \\
\quad \Gamma; \lfloor\langle \mathsf{spawn}(\theta, \mathsf{e}, \mathrm{p}'') : \pi, \mathrm{p}, (\theta', \mathsf{e}'), q\rangle\rfloor_{\Psi} \;\&\; \langle \pi'', \mathrm{p}'', (\theta'', \mathsf{e}''), q'')\rangle \;\&\; \Pi \\
\qquad\qquad\qquad \leftharpoondown \Gamma; \lfloor\langle \pi, \mathrm{p}, (\theta, \mathsf{e}), q\rangle\rfloor_{\Psi} \;\&\; \lfloor\langle \pi'', \mathrm{p}'', (\theta'', \mathsf{e}''), q'')\rangle\rfloor_{\#^{\mathrm{p}''}_{\mathsf{sp}}} \;\&\; \Pi
\end{array}
$$

Backward semantics (3)

The rollback terminates when all checkpoints are reached
(and forward computation is resumed)

$$(\mathit{Check}) \quad \Gamma; \lfloor \langle \#^{\mathtt{t}}_{\mathsf{ch}} : \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\Psi \cup \#^{\mathtt{t}}_{\mathsf{ch}}} \mathbin{\&} \Pi \leftharpoondown \Gamma; \lfloor \langle \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\Psi} \mathbin{\&} \Pi$$

$$(\mathit{Stop1}) \quad \Gamma; \lfloor \langle \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\emptyset} \mathbin{\&} \Pi \leftharpoondown \Gamma; \langle \pi, \mathrm{p}, (\theta, e), q \rangle \mathbin{\&} \Pi$$

Conclusions

Conclusions and future work

We have defined a **reversible semantics for a subset of Erlang**

Ongoing work:

- correctness:
  - every forward step can be reversed
  - every system reached with the backward semantics, could have been reached with forward semantics from the initial system
- introduce a notion of safe session
- implementation

Conclusions and future work

We have defined a **reversible semantics for a subset of Erlang**

Ongoing work:

- correctness:
  - every forward step can be reversed
  - every system reached with the backward semantics, could have been reached with forward semantics from the initial system
- introduce a notion of safe session
- implementation

Conclusions and future work

We have defined a **reversible semantics for a subset of Erlang**

Ongoing work:

- correctness:
  - every forward step can be reversed
  - every system reached with the backward semantics, could have been reached with forward semantics from the initial system
- introduce a notion of safe session
- implementation

Conclusions and future work

We have defined a **reversible semantics for a subset of Erlang**

Ongoing work:

- correctness:
  - every forward step can be reversed
  - every system reached with the backward semantics, could have been reached with forward semantics from the initial system
- introduce a notion of safe session
- implementation

Thanks for your attention!

Backward semantics: checkpoints

Ψ is the set of *pending* checkpoints

$$(\mathit{Undo1}) \quad \Gamma; \langle \pi, \mathrm{p}, (\theta, e), q \rangle \;\&\; \Pi \leftharpoondown \Gamma; \lfloor \langle \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\#^{\mathtt{t}}_{\mathsf{ch}}} \;\&\; \Pi$$

$$(\mathit{Undo2}) \quad \Gamma; \lfloor \langle \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\Psi} \;\&\; \Pi \leftharpoondown \Gamma; \lfloor \langle \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\Psi \cup \#^{\mathtt{t}}_{\mathsf{ch}}} \;\&\; \Pi$$

$$(\mathit{Stop1}) \quad \Gamma; \lfloor \langle \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\emptyset} \;\&\; \Pi \leftharpoondown \Gamma; \langle \pi, \mathrm{p}, (\theta, e), q \rangle \;\&\; \Pi$$

$$(\mathit{Stop2}) \quad \Gamma; \lfloor \langle [\,], \mathrm{p}, (\theta, e), q \rangle \rfloor_{\{\#^{\mathtt{t}}_{\mathsf{sp}}\}} \;\&\; \Pi \leftharpoondown \Gamma; \Pi$$

$$(\mathit{Check}) \quad \Gamma; \lfloor \langle \#^{\mathtt{t}}_{\mathsf{ch}} : \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\Psi \cup \#^{\mathtt{t}}_{\mathsf{ch}}} \;\&\; \Pi \leftharpoondown \Gamma; \lfloor \langle \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\Psi} \;\&\; \Pi$$

$$(\mathit{Discard}) \quad \Gamma; \lfloor \langle \#^{\mathtt{t}}_{\mathsf{ch}} : \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\Psi} \;\&\; \Pi \leftharpoondown \Gamma; \lfloor \langle \pi, \mathrm{p}, (\theta, e), q \rangle \rfloor_{\Psi} \;\&\; \Pi \quad \text{if } \#^{\mathtt{t}}_{\mathsf{ch}} \notin \Psi$$

Backward semantics: system rules

$$(\mathit{Internal}) \quad \Gamma; \lfloor\langle \tau(\theta, e) : \pi, p, (\theta', e'), q\rangle\rfloor_\Psi \;\&\; \Pi \leftharpoondown \Gamma; \lfloor\langle \pi, p, (\theta, e), q\rangle\rfloor_\Psi \;\&\; \Pi$$

$$(\mathit{Check}) \quad \Gamma; \lfloor\langle \mathsf{check}(\theta, e) : \pi, p, (\theta', e'), q\rangle\rfloor_\Psi \;\&\; \Pi \leftharpoondown \Gamma; \lfloor\langle \pi, p, (\theta, e), q\rangle\rfloor_\Psi \;\&\; \Pi$$

$$(\mathit{Self}) \quad \Gamma; \lfloor\langle \mathsf{self}(\theta, e) : \pi, \mathrm{p}, (\theta', e'), q\rangle\rfloor_\Psi \;\&\; \Pi \leftharpoondown \Gamma; \lfloor\langle \pi, \mathrm{p}, (\theta, e), q\rangle\rfloor_\Psi \;\&\; \Pi$$

$$(\mathit{Spawn}) \quad \begin{array}{r} \Gamma; \lfloor\langle \mathsf{spawn}(\theta, e, \mathrm{p}'') : \pi, \mathrm{p}, (\theta', e'), q\rangle\rfloor_\Psi \;\&\; \langle \pi'', \mathrm{p}'', (\theta'', e''), q''\rangle \;\&\; \Pi \\ \leftharpoondown \Gamma; \lfloor\langle \pi, \mathrm{p}, (\theta, e), q\rangle\rfloor_\Psi \;\&\; \lfloor\langle \pi'', \mathrm{p}'', (\theta'', e''), q''\rangle\rfloor_{\#^{\mathrm{p}''}_{\mathsf{sp}}} \;\&\; \Pi \end{array}$$

Backward semantics: system rules (cont.)

$$(\textit{Receive}) \quad \Gamma; \lfloor\langle \mathsf{rec}(\theta, e, q) : \pi, \mathrm{p}, (\theta', e'), q' \rangle\rfloor_\Psi \;\&\; \Pi \leftharpoondown \Gamma; \lfloor\langle \pi, \mathrm{p}, (\theta, e), q \rangle\rfloor_\Psi \;\&\; \Pi$$

$$(\textit{Send1}) \quad \begin{array}{r} \Gamma; \lfloor\langle \mathsf{send}(\mathrm{p}'', \theta, e, \mathtt{t}) : \pi, \mathrm{p}, (\theta', e'), q \rangle\rfloor_\Psi \;\&\; \Pi \leftharpoondown \Gamma'; \lfloor\langle \pi, \mathrm{p}, (\theta, e), q \rangle\rfloor_\Psi \;\&\; \Pi \\ \text{if } (\mathrm{p}, \mathrm{p}'', \{\mathtt{t}, v\}) \text{ occurs in } \Gamma, \text{ with } \Gamma' = \Gamma \setminus\!\!\setminus (\mathrm{p}, \mathrm{p}'', v, \mathtt{t}) \end{array}$$

$$(\textit{Send2}) \quad \begin{array}{r} \Gamma; \lfloor\langle \mathsf{send}(\mathrm{p}'', \theta, e, \mathtt{t}) : \pi, \mathrm{p}, (\theta', e'), q \rangle\rfloor_\Psi \;\&\; \langle \pi'', \mathrm{p}'', (\theta'', e''), q'' \rangle \;\&\; \Pi \\ \leftharpoondown \Gamma; \lfloor\langle \pi, \mathrm{p}, (\theta, e), q \rangle\rfloor_\Psi \;\&\; \lfloor\langle \pi'', \mathrm{p}'', (\theta'', e''), q'' \rangle\rfloor_{\#^{\mathtt{t}}_{\alpha}} \;\&\; \Pi \\ \text{if } (\mathrm{p}, \mathrm{p}'', \{\mathtt{t}, v\}) \text{ does not occur in } \Gamma \end{array}$$

$$(\textit{Sched1}) \quad \Gamma; \lfloor\langle \alpha(\mathrm{p}'', \mathrm{p}, \{\mathtt{t}, v\}) : \pi, \mathrm{p}, (\theta, e), \{\mathtt{t}, v\} : q \rangle\rfloor_{\Psi \cup \#^{\mathtt{t}}_{\alpha}} \;\&\; \Pi \leftharpoondown \Gamma; \lfloor\langle \pi, \mathrm{p}, (\theta, e), q \rangle\rfloor_\Psi \;\&\; \Pi$$

$$(\textit{Sched2}) \quad \begin{array}{r} \Gamma; \lfloor\langle \alpha(\mathrm{p}'', \mathrm{p}, \{\mathtt{t}, v\}) : \pi, \mathrm{p}, (\theta, e), \{\mathtt{t}, v\} : q \rangle\rfloor_\Psi \;\&\; \Pi \\ \leftharpoondown \Gamma \,\overline{\cup}\, (\mathrm{p}'', \mathrm{p}, \{\mathtt{t}, v\}); \lfloor\langle \pi, \mathrm{p}, (\theta, e), q \rangle\rfloor_\Psi \;\&\; \Pi \quad \text{if } \#^{\mathtt{t}}_{\alpha} \notin \Psi \end{array}$$