A Semantics of Core Erlang With Handling of Signals

Aurélie Kong Win Chang
Jérôme Feret
Gregor Gössler

Inria – Projet ANR Dcore

Motivations and Context

Fault explanation in order to debug

- bugs caused by non determinism
- focus on evaluation order of messages

Needs

- a small step semantics
- describing symetrical links and monitoring links
- handling signals

→ **Problem:** no existing semantics cover these mechanisms

Design Choices

Erlang system

- hierarchy of concurrent processes
- interactions only via asynchronous signals
- no nodes (simplification)
- no global variables (simplification)
- no continuous time (simplification)

Rules of thumb

- the official documentation and initial specification first
- if an information is needed but absent from the documentation, go to the implementation
- if documentation and implementation disagree, the documentation wins

Syntax

Choices we made

- Core Erlang – less syntactic sugar
- a restored `receive`, but without the `after` clause
- no `try_catch` structure

Execution model

State of a system

$$\Pi \cup \{(pid, (e, \theta, \tau, m, stack, r), sig_{sent}, L)\}$$

- Π = set of the others processes

Execution model

State of a system

$$\Pi \cup \{(\textbf{pid}, (e, \theta, \tau, m, stack, r), sig_{sent}, L)\}$$

- pid = process identifier

Execution model

State of a system – execution task

$$\Pi \cup \{(pid, (\mathbf{e}, \theta, \tau, \mathbf{m}, \mathbf{stack}, \mathbf{r}), sig_{sent}, L)\}$$

- Expression evaluation:
  - e = current expression
  - θ = evaluation context
  - τ = functions table
  - m = current module
- Debuging info:
  - $stack$ = current call stack
  - r = ending reason

Execution model

State of a system – signals

$$\Pi \cup \{(pid, (e, \theta, \tau, m, stack, r), \mathbf{sig_{sent}}, L)\}$$

- outbox: list of signals sent by pid, in the order of their emission
- two kinds of signals:
  - messages – treated during `receive` expression
  - others – treated as soon as they are received
- general form of a signal: $(flag_{type}, pid_{target}, content)$

Execution model

State of a system – signals

$$\Pi \cup \{(pid, (e, \theta, \tau, m, stack, r), \mathbf{sig_{sent}}, L)\}$$

Guarantees on the order:

- if a process sends signals to another one, the reception order is the same as the sending order
- if more than one process send signals to a same process, no guarantee on the order of signals coming from different processes

Execution model

State of a system – links

$$\Pi \cup \{(pid, (e, \theta, \tau, m, stack, r), sig_{sent}, \mathbf{L})\}$$

- set of links
- two kinds of links:
  - monitor links
    - monitored side: $(\texttt{monitored\_by}, pid_i, lid)$
    - monitoring side: $(\texttt{monitoring}, pid_i, lid)$
  - symetrical link: $(\texttt{link}, pid_i, pid_j)$

Semantics – Running Example

In the file of the 'test' module

```
'go'/0 = fun() ->
  do call 'erlang':'+'(2,3)
     call 'erlang':'+'(4,4)
```

First instruction

```
do
 call 'erlang':'monitor'(process,
   call 'erlang':'spawn'('test','go',[])
 )
 receive
   _ when 'true' -> 'stop'
```

Semantics – Running Example

In the file of the 'test' module

```
'go'/0 = fun() ->
  do call 'erlang':'+'(2,3)
     call 'erlang':'+'(4,4)
```

First instruction

```
do
 call 'erlang':'monitor'(process,
   call 'erlang':'spawn'('test','go',[])
 )
 receive
   _ when 'true' -> 'stop'
```

parent:

- creates a child with first instruction `call test:go()`
- creates a monitoring link with its child
- waits for a message

Semantics – Running Example

In the file of the 'test' module

```
'go'/0 = fun() ->
  do call 'erlang':'+'(2,3)
     call 'erlang':'+'(4,4)
```

First instruction

```
do
 call 'erlang':'monitor'(process,
   call 'erlang':'spawn'('test','go',[])
  )
 receive
   _ when 'true' -> 'stop'
```

parent:

- creates a child with first instruction `call test:go()`
- creates a monitoring link with its child
- waits for a message

Semantics – Running Example

In the file of the 'test' module

```
'go'/0 = fun() ->
  do call 'erlang':'+'(2,3)
     call 'erlang':'+'(4,4)
```

First instruction

```
do
 call 'erlang':'monitor'(process,
   call 'erlang':'spawn'('test','go',[])
 )
 receive
   _ when 'true' -> 'stop'
```

parent:

- creates a child with first instruction `call test:go()`
- creates a monitoring link with its child
- waits for a message

Semantics – Running Example

In the file of the 'test' module

```
'go'/0 = fun() ->
  do call 'erlang':'+'(2,3)
     call 'erlang':'+'(4,4)
```

First instruction

```
do
 call 'erlang':'monitor'(process,
   call 'erlang':'spawn'('test','go',[])
 )
 receive
   _ when 'true' -> 'stop'
```

child:

- evaluates
  `call erlang:+(2,3)`
- receives the monitoring signal from parent
- updates L
- evaluates
  `call erlang:+(4,4)`
- terminates

Semantics – Running Example

In the file of the 'test' module

```
'go'/0 = fun() ->
  do call 'erlang':'+'(2,3)
     call 'erlang':'+'(4,4)
```

First instruction

```
do
 call 'erlang':'monitor'(process,
   call 'erlang':'spawn'('test','go',[])
 )
 receive
   _ when 'true' -> 'stop'
```

child:

- evaluates
  `call erlang:+(2,3)`
- receives the monitoring signal from parent
- updates L
- evaluates
  `call erlang:+(4,4)`
- terminates

Semantics – Running Example

In the file of the 'test' module

```
'go'/0 = fun() ->
  do call 'erlang':'+'(2,3)
     call 'erlang':'+'(4,4)
```

First instruction

```
do
 call 'erlang':'monitor'(process,
   call 'erlang':'spawn'('test','go',[])
 )
 receive
   _ when 'true' -> 'stop'
```

child:

- evaluates `call erlang:+(2,3)`
- receives the monitoring signal from parent
- updates L
- evaluates `call erlang:+(4,4)`
- terminates

Semantics – Running Example

In the file of the 'test' module

```
'go'/0 = fun() ->
  do call 'erlang':'+'(2,3)
     call 'erlang':'+'(4,4)
```

First instruction

```
do
 call 'erlang':'monitor'(process,
   call 'erlang':'spawn'('test','go',[])
 )
 receive
   _ when 'true' -> 'stop'
```

child:

- evaluates `call erlang:+(2,3)`
- receives the monitoring signal from parent
- updates L
- evaluates `call erlang:+(4,4)`
- terminates

Semantics – Running Example

In the file of the 'test' module

```
'go'/0 = fun() ->
  do call 'erlang':'+'(2,3)
     call 'erlang':'+'(4,4)
```

First instruction

```
do
 call 'erlang':'monitor'(process,
   call 'erlang':'spawn'('test','go',[])
 )
 receive
   _ when 'true' -> 'stop'
```

child:

- evaluates `call erlang:+(2,3)`
- receives the monitoring signal from parent
- updates L
- evaluates `call erlang:+(4,4)`
- terminates

Semantics – Running Example

In the file of the 'test' module

```
'go'/0 = fun() ->
  do call 'erlang':'+'(2,3)
     call 'erlang':'+'(4,4)
```

First instruction

```
do
 call 'erlang':'monitor'(process,
   call 'erlang':'spawn'('test','go',[])
 )
 receive
   _ when 'true' -> 'stop'
```

parent:

- receives the down signal from child
- turns the down signal into a message
- follows the clause of its receive
- terminates

Semantics – Running Example

In the file of the 'test' module

```
'go'/0 = fun() ->
  do call 'erlang':'+'(2,3)
     call 'erlang':'+'(4,4)
```

First instruction

```
do
 call 'erlang':'monitor'(process,
   call 'erlang':'spawn'('test','go',[])
 )
 receive
   _ when 'true' -> 'stop'
```

parent:

- receives the down signal from child
- turns the down signal into a message
- follows the clause of its receive
- terminates

Semantics – Running Example

In the file of the 'test' module

```
'go'/0 = fun() ->
  do call 'erlang':'+'(2,3)
     call 'erlang':'+'(4,4)
```

First instruction

```
do
 call 'erlang':'monitor'(process,
   call 'erlang':'spawn'('test','go',[])
 )
 receive
   _ when 'true' -> 'stop'
```

parent:

- receives the down signal from child
- turns the down signal into a message
- follows the clause of its receive
- terminates

Semantics – Running Example

In the file of the 'test' module

```
'go'/0 = fun() ->
  do call 'erlang':'+'(2,3)
     call 'erlang':'+'(4,4)
```

First instruction

```
do
 call 'erlang':'monitor'(process,
   call 'erlang':'spawn'('test','go',[])
 )
 receive
   _ when 'true' -> 'stop'
```

parent:

- receives the down signal from child
- turns the down signal into a message
- follows the clause of its receive
- terminates

Semantics – End of a process I

Behavior

- Reasons of Termination:
  - normal evaluation of its first expression
  - a problem led to the end of the process
- Result:
  - sends a `down` signal for each monitoring or symetrical link
  - terminates itself

Semantically

- Two special values:
  - EoP: the process is about to be terminated
  - ended: the process is dead
- Two types of transitions:
  - $\xrightarrow{term}$: evaluation of expressions at top level
  - $\xrightarrow{aux}$: exploration and evaluation of sub-expressions recursively

$\rightarrow$ Necessary to handle signal of end of processes

Semantics – End of a process II

From *aux* to *term*

$$\text{EXPR\_TERM}\ \frac{\Pi \cup \{(pid, (e_1, \theta, \tau, \mathtt{m}, stack \cdot (e_1, \theta, \mathtt{m}), \mathtt{r}), s_{sent}, L)\} \xrightarrow{aux} \Pi' \cup \{(pid, (e_1', \theta', \tau', \mathtt{m}', stack, \mathtt{r}'), s_{sent}', L')\}}{\Pi \cup \{(pid, (e_1, \theta, \tau, \mathtt{m}, stack, \mathtt{r}), s_{sent}, L)\} \xrightarrow{term} \Pi' \cup \{(pid, (e_1', \theta, \tau', \mathtt{m}', stack, \mathtt{r}'), s_{sent}', L')\}}$$

Semantics – End of a process III

Last evaluation: sending signals

$$\textsc{Val}\ \frac{\begin{array}{c} J = \{i \mid 1 \leqslant i \leqslant n \wedge fl_i \in \{\texttt{monitored\_by}, \texttt{link}\}\} \\ M = \{(\texttt{not\_msg}, pid_j, (\texttt{down}, fl_j, lid_j, \text{end\_reason}(\texttt{r}, normal))) \mid j \in J\} \\ sigs \in \text{permutations}(M) \qquad v \neq ended \end{array}}{\begin{array}{c} \Pi \cup \\ \{(pid, (v, \theta, \tau, \texttt{m}, stack, \texttt{r}), sig_{sent}, \{(fl_i, pid_i, lid_i) | 1 \leqslant i \leqslant n \wedge lid_i \in \{\mathcal{I}d, Ref_L\}\})\} \\ \xrightarrow{term} \\ \Pi \cup \\ \{(pid, (ended, \theta, \tau, \texttt{m}, stack, \texttt{r}), sig_{sent} \cdot sigs, \{(fl_i, pid_i, lid_i) | 1 \leqslant i \leqslant n \wedge lid_i \in \{\mathcal{I}d, Ref_L\}\})\} \end{array}}$$

Semantics – End of a process – Example

Child process: initial state

- born from `call 'erlang':'spawn'('test', 'go', [])`
- first instruction: `call 'test':'go'()`

$$\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\}$$

Semantics – End of a process – Example

Child process: initial state

- born from `call 'erlang':'spawn'('test', 'go', [])`
- first instruction: `call 'test':'go'()`

$$\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\}$$

pid of the child process

Semantics – End of a process – Example

Child process: initial state

- born from `call 'erlang':'spawn'('test', 'go', [])`
- first instruction: `call 'test':'go'()`

$$\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\}$$

first instruction

Semantics – End of a process – Example

Child process: initial state

- born from `call 'erlang':'spawn'('test', 'go', [])`
- first instruction: `call 'test':'go'()`

$$\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\}$$

empty evaluation context θ and initial functions table τ

Semantics – End of a process – Example

Child process: initial state

- born from `call 'erlang':'spawn'('test', 'go', [])`
- first instruction: `call 'test':'go'()`

$$\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\}$$

module: *test*, empty stack, reason of termination: $\bot$

Semantics – End of a process – Example

Child process: initial state

- born from `call 'erlang':'spawn'('test', 'go', [])`
- first instruction: `call 'test':'go'()`

$$\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\}$$

empty outbox

Semantics – End of a process – Example

Child process: initial state

- born from `call 'erlang':'spawn'('test', 'go', [])`
- first instruction: `call 'test':'go'()`

$$\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\}$$

empty set of links with other processes

Semantics – End of a process – Example

Child process: evaluation

- expression $\notin$ Val
- need to use $\xrightarrow{aux}$

$$\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, ((call\ test : go(), [], test)), \bot), (), \{\})\}$$

$$\xrightarrow{aux}$$

$$?$$

$$\text{EXPR\_\_TERM}\ \frac{}{\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\}}$$

$$\xrightarrow{term}$$

$$?$$

updated stack

Semantics – End of a process – Example

Child process: evaluation a few operations later

- expression $\notin$ Val
- need to use $\xrightarrow{aux}$

$$\text{EXPR\_\_TERM}\ \frac{\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, ((call\ \ test : go(), [], test)), \bot), (), \{\})\} \xrightarrow{aux} \Pi' \cup \{(pid, (8, [], \tau_{ini}, test, (), \bot), (), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\}}{\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\} \xrightarrow{term} \Pi' \cup \{(pid, (8, [], \tau_{ini}, test, (), \bot), (), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\}}$$

Semantics – End of a process – Example

Child process: evaluation a few operations later

- expression $\notin$ Val
- need to use $\xrightarrow{aux}$

$$\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, ((call\ test : go(), [], test)), \bot), (), \{\})\} \xrightarrow{aux} \Pi' \cup \{(pid, (8, [], \tau_{ini}, test, (), \bot), (), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\}$$

$$\text{EXPR\_\_TERM}\ \frac{\Pi' \cup \{(pid, (8, [], \tau_{ini}, test, (), \bot), (), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\}}{\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\}}$$

$$\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\} \xrightarrow{term} \Pi' \cup \{(pid, (\mathbf{8}, [], \tau_{ini}, test, (), \bot), (), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\}$$

resulting expression

Semantics – End of a process – Example

Child process: evaluation a few operations later

- expression $\notin$ Val
- need to use $\xrightarrow{aux}$

$$\text{EXPR\_\_TERM}\ \frac{\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, ((call\ test : go(), [], test)), \bot), (), \{\})\} \xrightarrow{aux} \Pi' \cup \{(pid, (8, [], \tau_{ini}, test, (), \bot), (), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\}}{\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\} \xrightarrow{term} \Pi' \cup \{(pid, (8, [], \tau_{ini}, test, (), \bot), (), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\}}$$

normal termination: reason of termination $= \bot$

Semantics – End of a process – Example

Child process: evaluation a few operations later

- expression $\notin$ Val
- need to use $\xrightarrow{aux}$

$$\text{EXPR\_\_TERM}\ \frac{\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, ((call\ test : go(), [], test)), \bot), (), \{\})\} \xrightarrow{aux} \Pi' \cup \{(pid, (8, [], \tau_{ini}, test, (), \bot), (), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\}}{\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\} \xrightarrow{term} \Pi' \cup \{(pid, (8, [], \tau_{ini}, test, (), \bot), \mathbf{()}, \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\}}$$

no signal sent: empty outbox

Semantics – End of a process – Example

Child process: evaluation a few operations later

- expression $\notin$ Val
- need to use $\xrightarrow{aux}$

$$\text{EXPR\_\_TERM}\ \frac{\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, ((call\ test : go(), [], test)), \bot), (), \{\})\} \xrightarrow{aux} \Pi' \cup \{(pid, (8, [], \tau_{ini}, test, (), \bot), (), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\}}{\Pi \cup \{(pid_{child}, (\underline{\texttt{call}}\ 'test' : go(), [], \tau_{ini}, test, (), \bot), (), \{\})\} \xrightarrow{term} \Pi' \cup \{(pid, (8, [], \tau_{ini}, test, (), \bot), (), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\}}$$

monitor link with the parent process

Semantics – End of a process – Example

Child process: terminating

- expression $\in$ Val
- can apply VAL

$$\textsc{Val}\ \frac{\begin{array}{c} M = \{(\texttt{not\_msg}, \mathit{pid}_{\mathit{parent}}, (\texttt{down}, \texttt{monitored\_by}, \mathit{lid}, \mathit{normal}))\} \\ \mathit{sigs} \in \text{permutations}(M) \qquad\qquad v \neq \mathit{ended} \end{array}}{\begin{array}{c} \Pi' \cup \{(\mathit{pid}, (8, [], \tau_{\mathit{ini}}, \mathit{test}, (), \bot), (), \{(\texttt{monitored\_by}, \mathit{pid}_{\mathit{parent}}, \mathit{lid})\})\} \\ \xrightarrow{\mathit{term}} \\ \Pi' \cup \\ \{(\mathit{pid}, (\mathit{ended}, [], \tau_{\mathit{ini}}, \mathit{test}, (), \bot), ((\texttt{not\_msg}, \mathit{pid}_{\mathit{parent}}, (\texttt{down}, \texttt{monitored\_by}, \mathit{lid}, \mathit{normal}))), \\ \{(\texttt{monitored\_by}, \mathit{pid}_{\mathit{parent}}, \mathit{lid})\})\} \end{array}}$$

progression toward *ended*

Semantics – End of a process – Example

Child process: terminating

- expression $\in$ Val
- can apply VAL

$$\textsc{Val}\ \frac{\begin{array}{c} M = \{(\texttt{not\_msg}, \mathit{pid}_{\mathit{parent}}, (\texttt{down}, \texttt{monitored\_by}, \mathit{lid}, \mathit{normal}))\} \\ \mathit{sigs} \in \text{permutations}(M) \qquad\qquad v \neq \mathit{ended} \end{array}}{\begin{array}{c} \Pi' \cup \{(\mathit{pid}, (8, [], \tau_{\mathit{ini}}, \mathit{test}, (), \bot), (), \{(\texttt{monitored\_by}, \mathit{pid}_{\mathit{parent}}, \mathit{lid})\})\} \\ \xrightarrow{\mathit{term}} \\ \Pi' \cup \\ \{(\mathit{pid}, (\mathit{ended}, [], \tau_{\mathit{ini}}, \mathit{test}, (), \bot), ((\texttt{not\_msg}, \mathit{pid}_{\mathit{parent}}, (\texttt{down}, \texttt{monitored\_by}, \mathit{lid}, \mathit{normal}))), \\ \{(\texttt{monitored\_by}, \mathit{pid}_{\mathit{parent}}, \mathit{lid})\})\} \end{array}}$$

one link flagged "`monitored_by `" $\rightarrow$ one signal sent

Semantics – End of a process – Example

Child process: terminating

- expression $\in$ Val
- can apply VAL

$$\textsc{Val}\ \frac{\begin{array}{c} M = \{(\texttt{not\_msg}, \mathit{pid}_{\mathit{parent}}, (\texttt{down}, \texttt{monitored\_by}, \mathit{lid}, \mathit{normal}))\} \\ \mathit{sigs} \in \text{permutations}(M) \qquad v \neq \mathit{ended} \end{array}}{\Pi' \cup \{(\mathit{pid}, (8, [], \tau_{\mathit{ini}}, \mathit{test}, (), \bot), (), \{(\texttt{monitored\_by}, \mathit{pid}_{\mathit{parent}}, \mathit{lid})\})\} \xrightarrow{\mathit{term}} \Pi' \cup \{(\mathit{pid}, (\mathit{ended}, [], \tau_{\mathit{ini}}, \mathit{test}, (), \bot), ((\texttt{not\_msg}, \mathit{pid}_{\mathit{parent}}, (\texttt{down}, \texttt{monitored\_by}, \mathit{lid}, \mathit{normal}))), \{(\texttt{monitored\_by}, \mathit{pid}_{\mathit{parent}}, \mathit{lid})\})\}}$$

signal: information about the signal (nature, destination)

Semantics – End of a process – Example

Child process: terminating

- expression $\in$ Val
- can apply VAL

$$\textsc{Val}\ \frac{\begin{array}{c} M = \{(\texttt{not\_msg}, \mathit{pid}_{\mathit{parent}}, (\texttt{down}, \texttt{monitored\_by}, \mathit{lid}, \mathit{normal}))\} \\ \mathit{sigs} \in \text{permutations}(M) \qquad v \neq \mathit{ended} \end{array}}{\begin{array}{c} \Pi' \cup \{(\mathit{pid}, (8, [], \tau_{\mathit{ini}}, \mathit{test}, (), \bot), (), \{(\texttt{monitored\_by}, \mathit{pid}_{\mathit{parent}}, \mathit{lid})\})\} \\ \xrightarrow{\mathit{term}} \\ \Pi' \cup \\ \{(\mathit{pid}, (\mathit{ended}, [], \tau_{\mathit{ini}}, \mathit{test}, (), \bot), ((\texttt{not\_msg}, \mathit{pid}_{\mathit{parent}}, (\texttt{down}, \texttt{monitored\_by}, \mathit{lid}, \mathit{normal}))), \\ \{(\texttt{monitored\_by}, \mathit{pid}_{\mathit{parent}}, \mathit{lid})\})\} \end{array}}$$

signal: information about the link (nature, identifier)

Semantics – End of a process – Example

Child process: terminating

- expression $\in$ Val
- can apply VAL

$$\textsc{Val}\ \frac{\begin{array}{c} M = \{(\texttt{not\_msg}, \mathit{pid}_{\mathit{parent}}, (\texttt{down}, \texttt{monitored\_by}, \mathit{lid}, \mathit{normal}))\} \\ \mathit{sigs} \in \text{permutations}(M) \qquad\qquad v \neq \mathit{ended} \end{array}}{\begin{array}{c} \Pi' \cup \{(\mathit{pid}, (8, [], \tau_{\mathit{ini}}, \mathit{test}, (), \bot), (), \{(\texttt{monitored\_by}, \mathit{pid}_{\mathit{parent}}, \mathit{lid})\})\} \\ \xrightarrow{\mathit{term}} \\ \Pi' \cup \\ \{(\mathit{pid}, (\mathit{ended}, [], \tau_{\mathit{ini}}, \mathit{test}, (), \bot), ((\texttt{not\_msg}, \mathit{pid}_{\mathit{parent}}, (\texttt{down}, \texttt{monitored\_by}, \mathit{lid}, \mathit{normal}))), \\ \{(\texttt{monitored\_by}, \mathit{pid}_{\mathit{parent}}, \mathit{lid})\})\} \end{array}}$$

signal: reason of termination

Semantics – Handling signals I

Characteristics

- Asynchronous
- Automatic: handled by default, impossible to prevent it

Down signals – monitor link

$$\text{SIG\_\_EXIT\_\_MONIT}\quad \frac{\begin{array}{c}\text{first\_sig}(sig_{sent,i}, j) = k \qquad i \neq j \\ s_{i,k} = (\texttt{not\_msg}, pid_j, (\texttt{down}, \texttt{monitored\_by}, Ref, r)) \\ (\texttt{monitoring}, pid_i, ref) \in L_j \qquad e_j \notin \{EoP, ended\}\end{array}}{\begin{array}{c}\Pi \cup \{(pid_j, (e_j, \theta_j, \tau_j, \mathtt{m}_j, stack_j, \mathtt{r}_j), sig_{sent,j}, L_j); \\ (pid_i, (e_i, \theta_i, \tau_i, \mathtt{m}_i, stack_i, \mathtt{r}_i), (s_{i,1}, \ldots, s_{i,k}, \ldots, s_{i,n_i}), L_i)\} \\ \xrightarrow{aux} \\ \Pi \cup \{(pid_j, (e_j, \theta_j, \tau_j, \mathtt{m}_j, stack_j, \mathtt{r}_j), sig_{sent,j}, L_j); \\ (pid_i, (e_i, \theta_i, \tau_i, \mathtt{m}_i, stack_i, \mathtt{r}_i), (s_{i,1}, \ldots, s_{i,k-1}, s_{i,k+1}, \ldots, s_{i,n_i}) \cdot \\ (\texttt{msg}, pid_j, (\{'DOWN', ref, \texttt{process}, pid_i, r\})), L_i)\}\end{array}}$$

Semantics – Handling signals II

Down signals – monitor link

$$\text{SIG\_\_EXIT\_\_MONIT}\quad \frac{\begin{array}{c}\text{first\_sig}(sig_{sent,i}, j) = k \qquad i \neq j \\ s_{i,k} = (\texttt{not\_msg}, pid_j, (\texttt{down}, \texttt{monitored\_by}, Ref, r)) \\ (\texttt{monitoring}, pid_i, ref) \in L_j \qquad e_j \notin \{EoP, ended\}\end{array}}{\begin{array}{c}\Pi \cup \{(pid_j, (e_j, \theta_j, \tau_j, \mathtt{m}_j, stack_j, \mathtt{r}_j), sig_{sent,j}, L_j); \\ (pid_i, (e_i, \theta_i, \tau_i, \mathtt{m}_i, stack_i, \mathtt{r}_i), (s_{i,1}, \ldots, s_{i,k}, \ldots, s_{i,n_i}), L_i)\} \\ \xrightarrow{aux} \\ \Pi \cup \{(pid_j, (e_j, \theta_j, \tau_j, \mathtt{m}_j, stack_j, \mathtt{r}_j), sig_{sent,j}, L_j); \\ (pid_i, (e_i, \theta_i, \tau_i, \mathtt{m}_i, stack_i, \mathtt{r}_i), (s_{i,1}, \ldots, s_{i,k-1}, s_{i,k+1}, \ldots, s_{i,n_i}) \cdot \\ (\mathtt{msg}, pid_j, (\{'DOWN', ref, \mathtt{process}, pid_i, r\})), L_i)\}\end{array}}$$

two processes

Semantics – Handling signals II

Down signals – monitor link

$$\text{SIG\_\_EXIT\_\_MONIT}\quad \frac{\begin{array}{c}\text{first\_sig}(sig_{sent,i}, j) = k \qquad i \neq j \\ s_{i,k} = (\texttt{not\_msg}, pid_j, (\texttt{down}, \texttt{monitored\_by}, Ref, r)) \\ (\texttt{monitoring}, pid_i, ref) \in L_j \qquad e_j \notin \{EoP, ended\}\end{array}}{\begin{array}{c}\Pi \cup \{(pid_j, (e_j, \theta_j, \tau_j, \mathtt{m}_j, stack_j, \mathtt{r}_j), sig_{sent,j}, L_j); \\ (pid_i, (e_i, \theta_i, \tau_i, \mathtt{m}_i, stack_i, \mathtt{r}_i), (s_{i,1}, \ldots, s_{i,k}, \ldots, s_{i,n_i}), L_i)\} \\ \xrightarrow{aux} \\ \Pi \cup \{(pid_j, (e_j, \theta_j, \tau_j, \mathtt{m}_j, stack_j, \mathtt{r}_j), sig_{sent,j}, L_j); \\ (pid_i, (e_i, \theta_i, \tau_i, \mathtt{m}_i, stack_i, \mathtt{r}_i), (s_{i,1}, \ldots, s_{i,k-1}, s_{i,k+1}, \ldots, s_{i,n_i}) \cdot \\ (\mathtt{msg}, pid_j, (\{'DOWN', ref, \mathtt{process}, pid_i, r\})), L_i)\}\end{array}}$$

the process j monitoring i has been sent a down signal

Semantics – Handling signals II

Down signals – monitor link

$$\text{SIG\_\_EXIT\_\_MONIT}\quad \frac{\begin{array}{c}\text{first\_sig}(sig_{sent,i}, j) = k \qquad i \neq j \\ s_{i,k} = (\texttt{not\_msg}, pid_j, (\texttt{down}, \texttt{monitored\_by}, Ref, r)) \\ (\texttt{monitoring}, pid_i, ref) \in L_j \qquad e_j \notin \{EoP, ended\}\end{array}}{\begin{array}{c}\Pi \cup \{(pid_j, (e_j, \theta_j, \tau_j, \mathtt{m}_j, stack_j, \mathtt{r}_j), sig_{sent,j}, L_j); \\ (pid_i, (e_i, \theta_i, \tau_i, \mathtt{m}_i, stack_i, \mathtt{r}_i), (s_{i,1}, \ldots, s_{i,k}, \ldots, s_{i,n_i}), L_i)\} \\ \xrightarrow{aux} \\ \Pi \cup \{(pid_j, (e_j, \theta_j, \tau_j, \mathtt{m}_j, stack_j, \mathtt{r}_j), sig_{sent,j}, L_j); \\ (pid_i, (e_i, \theta_i, \tau_i, \mathtt{m}_i, stack_i, \mathtt{r}_i), (s_{i,1}, \ldots, s_{i,k-1}, s_{i,k+1}, \ldots, s_{i,n_i}) \cdot \\ (\texttt{msg}, pid_j, (\{'DOWN', ref, \texttt{process}, pid_i, r\})), L_i)\}\end{array}}$$

signal deleted, corresponding message appended to the outbox

Semantics – Handling signals – Example

$$\text{\_\_EXIT\_\_MONIT}\ \frac{\begin{array}{c}\text{first\_sig}(((\texttt{not\_msg}, pid_{parent}, (\texttt{down}, \texttt{monitored\_by}, lid, normal))), parent) = 1 \\ s_{child,1} = (\texttt{not\_msg}, pid_{parent}, (\texttt{down}, \texttt{monitored\_by}, lid, normal)) \\ (\texttt{monitoring}, pid_{child}, lid) \in L_{parent} \qquad e_{parent} \notin \{EoP, ended\}\end{array}}{\begin{array}{c}\{(pid_{parent}, (\underline{\texttt{receive}}\ \_\ \underline{\texttt{when}}\ 'true' ->' stop', [], \tau_{parent}, erlang, \\ stack_{parent}, \bot), (), \{(\texttt{monitoring}, pid_{child}, lid)\}); \\ (pid_{child}, (ended, [], \tau_{ini}, test, (), \bot), ((\texttt{not\_msg}, pid_{parent}, (\texttt{down}, \\ \texttt{monitored\_by}, lid, normal))), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\} \\ \xrightarrow{aux} \\ \{(pid_{parent}, (\underline{\texttt{receive}}\ \_\ \underline{\texttt{when}}\ 'true' ->' stop', [], \tau_{parent}, erlang, \\ stack_{parent}, \bot), (), \{(\texttt{monitoring}, pid_{child}, lid)\}); \\ (pid_{child}, (ended, [], \tau_{ini}, test, (), \bot), \\ (\texttt{msg}, pid_{parent}, (\{'DOWN', lid, \texttt{process}, pid_{child}, normal\}), L_{child})\}\end{array}}$$

Semantics – Handling signals – Example

$$\text{first\_sig}(((\texttt{not\_msg}, pid_{parent}, (\texttt{down}, \texttt{monitored\_by}, lid, normal))), parent) = 1$$
$$s_{child,1} = (\texttt{not\_msg}, pid_{parent}, (\texttt{down}, \texttt{monitored\_by}, lid, normal))$$
$$(\texttt{monitoring}, pid_{child}, lid) \in L_{parent} \qquad e_{parent} \notin \{EoP, ended\}$$

__EXIT__MONIT

$$\{(pid_{parent}, (\underline{\texttt{receive}}\ \_\ \underline{\texttt{when}}\ 'true' - >' stop', [], \tau_{parent}, erlang, stack_{parent}, \bot), (), \{(\texttt{monitoring}, pid_{child}, lid)\});$$
$$(pid_{child}, (ended, [], \tau_{ini}, test, (), \bot), ((\texttt{not\_msg}, pid_{parent}, (\texttt{down}, \texttt{monitored\_by}, lid, normal))), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\}$$
$$\xrightarrow{aux}$$
$$\{(pid_{parent}, (\underline{\texttt{receive}}\ \_\ \underline{\texttt{when}}\ 'true' - >' stop', [], \tau_{parent}, erlang, stack_{parent}, \bot), (), \{(\texttt{monitoring}, pid_{child}, lid)\});$$
$$(pid_{child}, (ended, [], \tau_{ini}, test, (), \bot),$$
$$(\texttt{msg}, pid_{parent}, (\{'DOWN', lid, \texttt{process}, pid_{child}, normal\}), L_{child})\}$$

two processes, parent monitoring child

Semantics – Handling signals – Example

Down signals – monitor link

$$\text{EXIT\_MONIT}\ \frac{\begin{array}{c}\text{first\_sig}(((\texttt{not\_msg}, pid_{parent}, (\texttt{down}, \texttt{monitored\_by}, lid, normal))), parent) = 1 \\ s_{child,1} = (\texttt{not\_msg}, pid_{parent}, (\texttt{down}, \texttt{monitored\_by}, lid, normal)) \\ (\texttt{monitoring}, pid_{child}, lid) \in L_{parent} \qquad e_{parent} \notin \{EoP, ended\}\end{array}}{\begin{array}{c}\{(pid_{parent}, (\underline{\texttt{receive}}\ \_\ \underline{\texttt{when}}\ 'true' -{>}'\ stop', [], \tau_{parent}, erlang, \\ stack_{parent}, \bot), (), \{(\texttt{monitoring}, pid_{child}, lid)\}); \\ (pid_{child}, (ended, [], \tau_{ini}, test, (), \bot), ((\texttt{not\_msg}, pid_{parent}, (\texttt{down}, \\ \texttt{monitored\_by}, lid, normal))), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\} \\ \xrightarrow{aux} \\ \{(pid_{parent}, (\underline{\texttt{receive}}\ \_\ \underline{\texttt{when}}\ 'true' -{>}'\ stop', [], \tau_{parent}, erlang, \\ stack_{parent}, \bot), (), \{(\texttt{monitoring}, pid_{child}, lid)\}); \\ (pid_{child}, (ended, [], \tau_{ini}, test, (), \bot), \\ (\texttt{msg}, pid_{parent}, (\{'DOWN', lid, \texttt{process}, pid_{child}, normal\}), L_{child})\}\end{array}}$$

child sent a down signal to parent

Semantics – Handling signals – Example

Down signals – monitor link

$$\text{\_\_EXIT\_\_MONIT}\ \frac{\begin{array}{c}\text{first\_sig}(((\texttt{not\_msg}, pid_{parent}, (\texttt{down}, \texttt{monitored\_by}, lid, normal))), parent) = 1 \\ s_{child,1} = (\texttt{not\_msg}, pid_{parent}, (\texttt{down}, \texttt{monitored\_by}, lid, normal)) \\ (\texttt{monitoring}, pid_{child}, lid) \in L_{parent} \qquad e_{parent} \notin \{EoP, ended\}\end{array}}{\begin{array}{c}\{(pid_{parent}, (\underline{\texttt{receive}}\ \_\ \underline{\texttt{when}}\ 'true' -{>}'\ stop', [], \tau_{parent}, erlang, \\ stack_{parent}, \bot), (), \{(\texttt{monitoring}, pid_{child}, lid)\}); \\ (pid_{child}, (ended, [], \tau_{ini}, test, (), \bot), ((\texttt{not\_msg}, pid_{parent}, (\texttt{down}, \\ \texttt{monitored\_by}, lid, normal))), \{(\texttt{monitored\_by}, pid_{parent}, lid)\})\} \\ \xrightarrow{aux} \\ \{(pid_{parent}, (\underline{\texttt{receive}}\ \_\ \underline{\texttt{when}}\ 'true' -{>}'\ stop', [], \tau_{parent}, erlang, \\ stack_{parent}, \bot), (), \{(\texttt{monitoring}, pid_{child}, lid)\}); \\ (pid_{child}, (ended, [], \tau_{ini}, test, (), \bot), \\ (\texttt{msg}, pid_{parent}, (\{'DOWN', lid, \texttt{process}, pid_{child}, normal\}), L_{child})\}\end{array}}$$

signal deleted, corresponding message appended to the outbox

Conclusion

Contribution

Small step semantics of a subset of Core Erlang:

- handling of monitoring links and symetrical links
- handling of signals

Coming . . .

- implementation
- investigate the causal analysis of faults originating from non deterministic messages order

End

Thank you

Syntax

$$
\begin{array}{lll}
\mathit{Module} & ::= & \underline{\texttt{module}}\ \mathit{atom}\ [\mathit{Fname}_{i_1}, \ldots, \mathit{Fname}_{i_k}] \\
 & & \underline{\texttt{attributes}}\ [\mathit{atom}_1 = \mathit{Cst}_1, \ldots, \mathit{atom}_m = \mathit{Cst}_m] \\
 & & \mathit{Fdef}_1 \ldots \mathit{Fdef}_n \\
\mathit{Fdef} & ::= & \mathit{Fname} = \mathit{Fun} \\
\mathit{Fname} & ::= & \mathit{atom}/\mathit{integer} \\
\mathit{Cst}(c) & ::= & \mathit{Lit} \mid [\ \mathit{Cst}_1 \mid \mathit{Cst}_2\ ] \\
 & \mid & \{\ \mathit{Cst}_1\ , \ldots\ , \mathit{Cst}_n\ \} \mid \mathit{Fun} \\
\mathit{Val}(v) & ::= & \mathit{Cst} \mid\ <\ \mathit{Cst}_1, \ldots, \mathit{Cst}_k\ >\ \mid \mathit{EoP} \mid \mathit{ended} \\
\mathit{Lit} & ::= & \mathit{integer}|\mathit{float}|\mathit{atom}|\mathit{char}|\mathit{string}|\ [\ ] \\
\mathit{Fun} & ::= & \underline{\texttt{fun}}\ (\ \mathit{var}_1, \ldots, \mathit{var}_n\ )\ ->\ \mathit{Exprs} \\
\mathit{Exprs} & ::= & \mathit{Expr} \mid\ <\ \mathit{Expr}_1, \ldots, \mathit{Expr}_n\ > \\
\mathit{Expr} & ::= & \mathit{var} \mid \mathit{Fname} \mid \mathit{Lit} \mid \mathit{Fun} \\
 & \mid & [\ \mathit{Expr}_1, \ldots, \mathit{Expr}_n\ ] \mid \{\ \mathit{Expr}_1, \ldots, \mathit{Expr}_n\ \} \\
 & \mid & \underline{\texttt{let}}\ \mathit{Vars}\ =\ \mathit{Exprs}_1\ \underline{\texttt{in}}\ \mathit{Exprs}_2 \\
 & \mid & \underline{\texttt{do}}\ \mathit{Exprs}_1\ \mathit{Exprs}_2 \\
 & \mid & \underline{\texttt{letrec}}\ \mathit{Fdef}_1\ \ldots\ \mathit{Fdef}_n\ \underline{\texttt{in}}\ \mathit{Exprs} \\
 & \mid & \underline{\texttt{apply}}\ \mathit{Exprs}_0\ (\mathit{Exprs}_1,\ \ldots,\ \mathit{Exprs}_n\ ) \\
 & \mid & \underline{\texttt{call}}\ \mathit{Exprs}_1\ :\ \mathit{Exprs}_2\ (\ \mathit{Exprs}_3\ , \ldots,\ \mathit{Exprs}_{n+2}\ ) \\
 & \mid & \underline{\texttt{primop}}\ \mathit{atom}\ (\ \mathit{Exprs}_1,\ \ldots,\ \mathit{Exprs}_n\ ) \\
 & \mid & \underline{\texttt{case}}\ \mathit{Exprs}\ \underline{\texttt{of}}\ \mathit{Cls}_1\ \ldots\ \mathit{Cls}_n\ \underline{\texttt{end}} \\
 & \mid & \underline{\texttt{receive}}\ \mathit{Cls}_1\ \ldots\ \mathit{Cls}_n\ \underline{\texttt{after}}\ \mathit{Exprs}_1\ ->\ \mathit{Exprs}_2 \\
\mathit{Vars}(x) & ::= & \mathit{var} \mid\ <\ \mathit{var}_1, \ldots, \mathit{var}_n\ > \\
\mathit{Cls}(cl) & ::= & \mathit{Pats}\ \underline{\texttt{when}}\ \mathit{Exprs}_1\ ->\ \mathit{Exprs}_2 \\
\mathit{Pats} & ::= & \mathit{Pat} \mid\ <\ \mathit{Pat}_1, \ldots, \mathit{Pat}_n\ > \\
\mathit{Pat}(p) & ::= & \mathit{var} \mid \mathit{Lit} \mid [\ \mathit{Pat}_1 \mid \mathit{Pat}_2\ ] \\
 & \mid & \{\ \mathit{Pat}_1, \ldots, \mathit{Pat}_n\ \} \mid \mathit{var}\ =\ \mathit{Pat}
\end{array}
$$

Syntax – receive

```
letrec 'recv$^0'/0 =
 fun() ->
  let <var1, var2> = primop 'recv_peck_message'()
   in case var1 of
    <'true'> when 'true' ->
   case var2 of
    Cls1 ... Clsn
    (<Other> when 'true' ->
      do primop 'recv_next'()
      (apply 'recv_$^0'/0())
    )
   end
  (<'false'> when 'true' ->
   let var3 = primop 'recv_wait_timeout'(Expr1)
   in case var3 of
      <'true'> when 'true' -> Expr2
      (<'false'> when 'true' -> (apply 'recv$^0'/0 ())
   end)
  end
in (apply 'recv$^0'/0 ())
```

Semantics – Recursive calls I

Choice

- Documentation: the arguments of a function can be evaluated in any order
- Current implementation: left to right, depth first
- Past implementation: right to left, depth first

$\rightarrow$ we are following the documentation

$$\textsc{Subexpr\_ok}\ \frac{\begin{array}{c} e_{eval} \in \mathit{Expr} \backslash \mathit{Val} \wedge e'_{eval} \neq \mathit{EoP} \\ \Pi \cup \{(\mathit{pid}, (e_{eval}, \theta, \tau, \mathtt{m}, \mathit{stack} \cdot (e_{eval}, \theta, \mathtt{m}), \mathtt{r}), s_{sent}, L)\} \\ \xrightarrow{aux} \\ \Pi' \cup \{(\mathit{pid}, (e'_{eval}, \theta', \tau', \mathtt{m}', \mathit{stack}', \mathtt{r}'), s'_{sent}, L')\} \end{array}}{\begin{array}{c} \Pi \cup \{(\mathit{pid}, (c[e_{eval}], \theta, \tau, \mathtt{m}, \mathit{stack}, \mathtt{r}), s_{sent}, L)\} \\ \xrightarrow{aux} \\ \Pi' \cup \{(\mathit{pid}, (c[e'_{eval}], \theta, \tau', \mathtt{m}, \mathit{stack}, \mathtt{r}'), s'_{sent}, L')\} \end{array}}$$

Semantics – Recursive calls II

Choice

- Documentation: the arguments of a function can be evaluated in any order
- Current implementation: left to right, depth first
- Past implementation: right to left, depth first

$\rightarrow$ we are following the documentation

$$\textsc{Subexpr\_ko}\ \frac{\begin{array}{c} e_{eval} \in Expr \backslash Val \\ \Pi \cup \{(pid, (e_{eval}, \theta, \tau, \mathtt{m}, stack \cdot (e_{eval}, \theta, \mathtt{m}), \mathtt{r}), s_{sent}, L)\} \\ \xrightarrow{aux} \\ \Pi' \cup \{(pid, (EoP, \theta', \tau', \mathtt{m}', stack', \mathtt{r}'), s'_{sent}, L')\} \end{array}}{\begin{array}{c} \Pi \cup \{(pid, (c[e_{eval}], \theta, \tau, \mathtt{m}, stack, \mathtt{r}), s_{sent}, L)\} \\ \xrightarrow{aux} \\ \Pi' \cup \{(pid, (EoP, \theta, \tau', \mathtt{m}, stack, \mathtt{r}'), s'_{sent}, L')\} \end{array}}$$

Semantics – Spawn

Spawning a process

$$\textsc{Rspawn}\ \frac{\mathsf{fresh\_pid}(\Pi) = pid_{fils} \qquad Mname, v_1, \ldots, v_n \in Val}{\begin{array}{c}\Pi \cup \{(pid_{pere}, (\underline{\texttt{call}}\ \texttt{'erlang':'spawn'}\\ (Mname, Fname, [v_1, \ldots, v_n]), \theta, \tau, \mathtt{m}), sig_{sent}, L)\}\\ \xrightarrow{aux}\\ \Pi \cup \{(pid_{pere}, (pid_{fils}, \theta, \tau, \mathtt{m}), sig_{sent}, L);\\ (pid_{fils}, (\underline{\texttt{call}}\ Mname : Fname(v_1, \ldots, v_n), [\,], \tau_{stat}, Mname), [\,], [\,])\}\end{array}}$$

Semantics – Handling signals – down link

Characteristics

- Asynchronous
- Automatic: handled by default, impossible to prevent it

Down signals – symetrical link

$$\text{SIG\_\_EXIT\_\_LINK}\ \frac{\begin{array}{c}\text{first\_sig}((s_{i,1},\ldots,s_{i,k},\ldots,s_{i,n_i}),j)=k \qquad i\neq j \\ s_{i,k}=(\texttt{not\_msg},pid_j,(\texttt{down},\texttt{link},pid_i,r)) \\ (\texttt{link},pid_i,pid_j)\in L_j \qquad e_j\notin\{EoP,ended\}\end{array}}{\begin{array}{c}\Pi\cup\{(pid_j,(e_j,\theta_j,\tau_j,\mathtt{m}_j,stack_j,\mathtt{r}_j),sig_{sent,j},L_j); \\ (pid_i,(e_i,\theta_i,\tau_i,\mathtt{m}_i,stack_i,\mathtt{r}_i),(s_{i,1},\ldots,s_{i,k},\ldots,s_{i,n_i}),L_i)\} \\ \xrightarrow{aux} \\ \Pi\cup\{(pid_j,(EoP,\theta_j,\tau_j,\mathtt{m}_j,stack_j,\text{end\_reason}(\mathtt{r}_j,r)),sig_{sent,j},L_j); \\ (pid_i,(e_i,\theta_i,\tau_i,\mathtt{m}_i,stack_i,\mathtt{r}_i),(s_{i,1},\ldots,s_{i,k-1},s_{i,k+1},\ldots,s_{i,n_i}),L_i)\}\end{array}}$$

Semantics – Monitor I

First step

$$\text{MONITOR}\ \frac{\mathsf{fresh\_lid}(L_1) = \mathit{lid}}{\Pi \cup \{(\mathit{pid}_1, (\underline{\texttt{call}}\ \mathit{erlang} : \mathit{monitor}(\texttt{process}, \mathit{pid}_2), \theta, \tau, \texttt{m}, \mathit{stack}, \texttt{r}), \mathit{sig}_{sent}, L)\} \xrightarrow{aux} \Pi \cup \{(\mathit{pid}_1, (\mathit{lid}, \theta, \tau, \mathit{erlang}, \texttt{m}, \mathit{stack}, \texttt{r}), \mathit{sig}_{sent} \cdot (\texttt{not\_msg}, \mathit{pid}_2, (\texttt{monitored\_by}, \mathit{pid}_1, \mathit{lid})), L_1 \cup \{(\texttt{monitoring}, \mathit{pid}_2, \mathit{lid})\})\}}$$

Update on the other side

$$\text{ONITORED\_BY}\ \frac{\mathsf{first\_sig}((s_{i,1}, \ldots, s_{i,k} \ldots, s_{i,n_i}), j) = k \qquad e_j \notin \{\mathit{EoP}, \mathit{ended}\} \qquad s_{i,k} = (\texttt{not\_msg}, \mathit{pid}_j, (\texttt{monitored\_by}, \mathit{pid}_i, \mathit{lid})) \qquad i \neq j}{\Pi \cup \{(\mathit{pid}_j, (e_j, \theta_j, \tau_j, \texttt{m}_j, \mathit{stack}_j, \texttt{r}_j), \mathit{sig}_{sent,j}, L_j); (\mathit{pid}_i, (e_i, \theta_i, \tau_i, \texttt{m}_i, \mathit{stack}_i, \texttt{r}_i), (s_{i,1}, \ldots, s_{i,k}, \ldots, s_{i,n_i}), L_i)\} \xrightarrow{aux} \Pi \cup \{(\mathit{pid}_j, (e_j, \theta_j, \tau_j, \texttt{m}_j, \mathit{stack}_j, \texttt{r}_j), \mathit{sig}_{sent,j}, L_j \cup \{(\texttt{monitored\_by}, \mathit{pid}_i, \mathit{lid})\}); (\mathit{pid}_i, (e_i, \theta_i, \tau_i, \texttt{m}_i, \mathit{stack}_i, \texttt{r}_i), (s_{i,1}, \ldots, s_{i,k-1}, s_{i,k+1}, \ldots, s_{i,n_i}), L_i)\}}$$