A Process-local Semantics

In this section, we show and explain the reduction rules of the process-local semantics. First, we describe a number of notations and meta-level definitions.

- An ether is a function of that maps (target) process identifiers to a list of pairs, which contain (source) process identifier - signal pairs.
- $+\!\!+$ denotes the concatenation of lists.
- :: is used for appending an element to the front of a list.
- $\in, \notin$ determine whether a value can (not) be found in a list.
- *rem* removes all occurrences of a value from a list.
- *remFirst*$(target, \Delta)$ removes only the first element from the list (which is a source process identifier - signal pair) associated with the specified target process identifier from the ether (Δ). If there is no such element, it returns the value *None*.
- *map* transforms a list based on its first parameter which is a meta-level function.
- *tt* denotes meta-level logical true.
- *ff* denotes meta-level logical false.
- *convert* is used to convert object-level values to their meta-level counter pair and vica versa (used only for boolean and list values).
- *is_ match* determines whether a value matches a pattern.
- *match* results the (pattern variable - value) bindings from a successful pattern matching.
- *insert*$(\Delta, target, (source, signal))$ inserts a (source process identifier - signal) pair into the ether (Δ) by appending it to the end of the list associated with the target identifier. Its use together with *remFirst* preserve the order of the signals targeting the same process.

Next, we extend the syntax of frames (continuations) of the sequential semantics:

$$K ::= \cdots \mid \Box(e_1, \ldots, e_k) \mid v(\Box, e_2, \ldots, e_k) \mid \cdots \mid v(v_1, \ldots, v_{k-1}, \Box)$$

A process is either:

- an alive process, which consists of a frame stack, evaluable expression, message queue (mailbox), list of linked process identifiers, and a flag about trapping exit signals (denoted by (K, e, q, pl, b)).
- or a dead process, which is a list of process identifiers associated with a reason value.

The evaluation of the parameters of the build-in function (BIF) call $e(e_1, \ldots, e_k)$ are handled by the sequential semantics (see the formalisation for more details [6]). Next, we make a brief description of the process-local reduction rules. Note, that the process identifiers in the following rules are propagated from the inter-process semantics. First we detail signal arrival (Figure 1):

- Seq lifts the sequential frame-stack semantics to the process-local level.
- Msg describes message arrival. Whenever a message arrives, it is appended to the mailbox of the process.

$$\frac{\langle K, e\rangle \hookrightarrow \langle K', e'\rangle}{(K, e, q, pl, b) \overset{\tau}{\hookrightarrow} (K', e', q, pl, b)} \quad \text{(Seq)}$$

$$(K, e, q, pl, b) \xhookrightarrow{arr(\iota_1, \iota_2, msg(v))} (K, e, q +\!\!+ [v], pl, b) \quad \text{(Msg)}$$

$$\frac{(\iota_1 \neq \iota_2 \wedge b = \mathit{ff} \wedge v = \texttt{'normal'}) \vee (\iota_1 \notin pl \wedge b_e = \mathit{tt} \wedge \iota_1 \neq \iota_2)}{(K, e, q, pl, b) \xhookrightarrow{arr(\iota_1, \iota_2, exit(v, b_e))} (K, e, q, pl, b)} \quad \text{(ExitDrop)}$$

$$\frac{\begin{array}{c}(v = \texttt{'kill'} \wedge b_e = \mathit{ff} \wedge v' = \texttt{'killed'}) \vee \\ (b = \mathit{ff} \wedge v \neq \texttt{'normal'} \wedge v' = v \wedge (b_e = \mathit{true} \rightarrow \iota_1 \in pl)) \vee \\ (b = \mathit{ff} \wedge v = \texttt{'normal'} = v' \wedge \iota_1 = \iota_2)\end{array}}{(K, e, q, pl, b) \xhookrightarrow{arr(\iota_1, \iota_2, exit(v, b_e))} \mathit{map}\ (\lambda \iota \Rightarrow (\iota, v'))\ pl} \quad \text{(ExitTerm)}$$

$$\frac{b = \mathit{tt} \wedge ((b_e = \mathit{ff} \wedge v \neq \texttt{'kill'}) \vee (b_e = \mathit{tt} \wedge \iota_1 \in pl))}{(K, e, q, pl, b) \xhookrightarrow{arr(\iota_1, \iota_2, exit(v, b_e))} (K, e, q +\!\!+ [\{\texttt{'EXIT'}, \iota_1, v\}], pl, b)} \quad \text{(ExitConv)}$$

$$(K, e, q, pl, b) \xhookrightarrow{arr(\iota_1, \iota_2, link)} (K, e, q, \iota_1 :: pl, b) \quad \text{(LinkArr)}$$

$$(K, e, q, pl, b) \xhookrightarrow{arr(\iota_1, \iota_2, unlink)} (K, e, q, \mathit{rem}(\iota_1, pl), b) \quad \text{(UnlinkArr)}$$

$$(\texttt{'send'}(\iota_2, \Box) :: K, v, q, pl, b) \xhookrightarrow{send(\iota_1, \iota_2, msg(v))} (K, v, q, pl, b) \quad \text{(Send)}$$

$$(\texttt{'exit'}(\iota_2, \Box) :: K, v, q, pl, b) \xhookrightarrow{send(\iota_1, \iota_2, exit(v, \mathit{ff}))} (K, \texttt{'true'}, q, pl, b) \quad \text{(Exit)}$$

$$(\texttt{'link'}(\Box) :: K, \iota_2, q, pl, b) \xhookrightarrow{send(\iota_1, \iota_2, link)} (K, \texttt{'ok'}, q, \iota_2 :: pl, b) \quad \text{(Link)}$$

$$(\texttt{'unlink'}(\Box) :: K, \iota_2, q, pl, b) \xhookrightarrow{send(\iota_1, \iota_2, unlink)} (K, \texttt{'ok'}, q, \mathit{rem}(\iota_2, pl), b) \quad \text{(Unlink)}$$

$$(\iota_2, v) :: pl \xhookrightarrow{send(\iota_1, \iota_2, exit(v, \mathit{tt}))} pl \quad \text{(Dead)}$$

Figure 1: Process local semantics of signal sends and arrival

- ExitDrop describes when should an exit signal be dropped.
- ExitTerm describes when an exit signal terminates the process. The process becomes a dead process by pairing the exit reason with the linked process identifiers. When an exit signal was sent explicitly, and the reason was `'kill'`, it has to be converted to `'killed'` for the links.
- ExitConv describes when an exit signal should be converted to a message and appended at the end of the mailbox. This action can only occur, when the trapping exits flag of the process is set.
- LinkArr, UnlinkArr rules describe arrival of link and unlink signals. In the first case, a process identifier is added to the links of the process, while in the second case, all occurrences of the process identifier is removed from the links.

Thereafter, we describe signal sending (Figure 1):

- Send describes message sending. If `'send'` BIF is on the top of the frame stack with the target process identifier, and the message is evaluated to a value, a send action is emitted with the source, target identifiers, and the message value, while the expression is reduced to the message value.
- Exit describes explicitly sending an exit signal to a process. If `'exit'` BIF is on the top of the frame stack with the target process identifier, and the reason is evaluated to a value, an exit action is emitted with the source, target identifiers, and the exit reason value, while the expression is reduced to `'true'`. Note, that when sending an explicit exit signal, the link flag of the signal is always false.
- Link, Unlink rules both reduce the evaluable expression to `'ok'`. In the first case, a link signal is emitted with the source and target identifier, and the target is appended to the links of the process. In the second case, an unlink signal is emitted with the source and target identifier, and the target is removed from the links of the process.
- Dead describes the communication of a dead process. In this rule, the first item of the links of the dead process is removed while an exit signal is emitted to the target with the reason that is specified in this first item. Note, that the link flag of this exit signal is true, because this exit is sent through a link.

Finally, we also detail the rest of the process-local rules (Figure 2):

- Self receives the identifier of the process from the inter-process semantics, and evaluates the `'self'` BIF call to this identifier.
- Spawn describes process creation. The spawned process receives its identifier from the inter-process semantics, and this value will be the result of this rule. Note, that it is necessary, that the first parameter of the `'spawn'` is a function value, while the second is a correct, object-level parameter list (which is checked in the inter-process semantics).
- Receive describes message processing. With pattern matching, the first (oldest) message is selected from the mailbox of the process that matches any clause of the `receive` expression (if more clauses are applicable, the first is selected). The evaluation continues with the selected clause, substituted by the result (pattern variable - value) bindings.
- Flag describes when the process flag for trapping exit signals change. The result of this rule is the previous value of the flag.

$$(\square() :: K, \texttt{'self'}, q, pl, b) \xrightarrow{self(\iota)} (K, \iota, q, pl, b) \quad \text{(Self)}$$

$$\frac{f = \texttt{fun}\ f/k(x_1, \ldots, x_k) \to e}{(\texttt{'spawn'}(f, \square) :: K, vs, q, pl, b) \xrightarrow{spawn(\iota, f, vs)} (K, \iota, q, pl, b)} \quad \text{(Spawn)}$$

$$\frac{is\_match(p_i, v) \qquad \forall j < i : \neg is\_match(p_j, v)}{(K, \texttt{receive}\ p_1 \to e_1; \ldots p_k \to e_k\ \texttt{end}, q, pl, b) \xrightarrow{self(\iota)} (K, e_i[match(p_i, v)], remFirst(v, q), pl, b)} \quad \text{(Receive)}$$

$$(\texttt{'process\_flag'}(\texttt{'trap\_exit'}, \square) :: K, v, q, pl, b) \xrightarrow{flag} (K, convert(b), q, pl, convert(v)) \quad \text{(Flag)}$$

$$(\mathcal{Id}, v, q, pl, b) \overset{\Downarrow}{\hookrightarrow} map\ (\lambda\iota \Rightarrow (\iota, \texttt{'normal'}))\ pl \quad \text{(Term)}$$

$$(\texttt{'exit'}() :: K, v, q, pl, b) \overset{\Downarrow}{\hookrightarrow} map\ (\lambda\iota \Rightarrow (\iota, v))\ pl \quad \text{(ExitSelf)}$$

Figure 2: Process-local semantics (part 2)

- Term describes normal termination, that is there is no more continuations in the frame stack, and the evaluable expression is a value. The result is a dead process, which will send exit signals to its links with the reason `'normal'`.
- ExitSelf describes the call of the single-parameter (which is the reason value) `'exit'` BIF. It immediately terminates the process, and exit signals will be sent to the linked processes with the reason value. We note that when introducing exceptions in the future, this version of the exit signals can be caught by exception-handlers.

B Inter-process Semantics

In this section, we discuss the inter-process reduction rules for the semantics. The advantage of this formalisation, is that the dynamic semantics of the system is described by only 5 rules (by combining the rules from related work [10, 12, 14] with the same premises, but different actions), which resulted in shorter proofs.

The semantics in Figure 3 reduces nodes, which are an ether and a partial function that maps process identifiers to (alive or dead) processes. We use Π to denote nodes, and $\iota : p \parallel \Pi$ to add the process p with the identifier ι to Π (if ι existed in Π, its corresponding process will be replaced by p). The order of the composition with $\parallel$ is irrelevant when the composition contains pairwise different process identifiers. This property characterises that at one state of the node, multiple reduction steps could be taken by different processes.

We use $\Pi \setminus \iota$ to denote the removal of the process identifier ι from Π, that is ι will not be associated with any process.

In this section, we discuss the inter-process reduction rules in Figure 3.

$$\frac{p \xrightarrow{send(\iota_1,\iota_2,s)} p'}{(\Delta, \iota_1 : p \parallel \Pi) \xrightarrow{\iota_1:send(\iota_1,\iota_2,s)} (insert(\Delta, (\iota_1, s), \iota_2), \iota_1 : p' \parallel \Pi)} \quad \text{(NSend)}$$

$$\frac{p \xrightarrow{arr(\iota_1,\iota_2,s)} p' \qquad remFirst(\iota_2, \Delta) = Some\ (\iota_1, s)}{(\Delta, \iota_1 : p \parallel \Pi) \xrightarrow{\iota_1:arr(\iota_1,\iota_2,s)} \Delta', \iota_1 : p' \parallel \Pi)} \quad \text{(NArrive)}$$

$$(\Delta, \iota : [] \parallel \Pi) \xrightarrow{\iota:\Downarrow} (\Delta, \Pi \setminus \iota) \quad \text{(NTerm)}$$

$$\frac{p \xrightarrow{spawn(\iota_2,v,vs)} p' \qquad v = \texttt{fun}\ f/k(x_1, \ldots, x_k) \to e}{(\Delta, \iota_1 : p \parallel \Pi) \xrightarrow{\iota_1:spawn(\iota_2,v,vs)} (\Delta, \iota_2 : ([], \texttt{apply}\ v(convert(vs)), [], [], \mathit{ff}) \parallel \iota_1 : p' \parallel \Pi)} \quad \text{(NSpawn)}$$

$$\frac{p \xhookrightarrow{a} p' \qquad a \in \{self(\iota), \Downarrow, \tau, flag\} \cup \{\forall v : rec(v)\}}{(\Delta, \iota : p \parallel \Pi) \xrightarrow{\iota:a} (\Delta, \iota : p' \parallel \Pi)} \quad \text{(NOther)}$$

Figure 3: Formal semantics of communication between processes

- NSend describes message sending. While a process with the identifier ι is reduced by a emitting a send action, the contents of this action (target, source, and signal) are placed inside the ether in an order-preserving way.
- NArrive describes how an element is (nondeterministically) removed from the ether, and put inside the mailbox of the specified process.
- NTerm describes how a process identifier is freed. When a dead process has notified all of its links, its identifier is removed from the node.
- NSpawn describes the creation of a new process. The new process is assigned a non-used identifier, and it starts evaluating the function application described in the spawn action of the rule (note, that conversion from object-level to meta-level lists is needed). The initial configuration of the new process is the empty frame stack (continuation), empty mailbox, it has no links, and it does not trap exit signals.
- NOther describes the reduction in case of any other action, that is, this rule propagates these actions to process-local level.