Specifying Broadcast Communication in E-LOTOS

MONIKA KAPUS-KOLAR
Department of digital communications and networks
"Jožef Stefan" Institute
Jamova 39, 1001 Ljubljana
SLOVENIA
monika.kapus-kolar@ijs.si

Abstract: The only form of inter-process communication in E-LOTOS, a standard process-algebraic language for specification of concurrent and reactive systems, is multiway synchronization. In an earlier work, we demonstrated that through abstract interpretation of events, E-LOTOS can also support broadcasting. The present paper proposes a generalization.

Key-Words: concurrent systems, formal specification, E-LOTOS, synchronous broadcasting.

1 Introduction

Broadcasting is in many cases the most convenient form of inter-process communication. However, not all formal languages directly support specification of broadcasting. In [1], we demonstrated that in E-LOTOS [2, 3], a standard process-algebraic specification language, broadcasting can be specified indirectly. The present paper generalizes the method.

2 Specification Language

In this section, we briefly describe the employed types of E-LOTOS processes.

"**stop**" denotes inaction. "**null**" denotes successful termination, i.e. a special event δ. "**i**" specifies an anonymous internal process action.

A "$GP_1@P_2 : T[E]$", where P_1, P_2, T and E are optional, specifies an interaction of the specified process at gate G. Pattern P_1 specifies data transmission, reception and/or matching associated with the action. T is the expected type of the data. P_2 denotes the time at which the action is executed. E is an additional condition on P_1 and P_2.

A "**signal** $X(E)$", where E is optional, denotes issuing of a signal X carrying data E. If such a signal indicates an exception, its issuing should better be specified by a "**raise** $X(E)$", leading to blocking of the system.

A "**trap exception** $X_1(IPL_1)$ **is** B_1 **endexn** ... **exception** $X_n(IPL_n)$ **is** B_n **endexn exit** P **is** B_{n+1} **endexit in** B **endtrap**", where the "**exit** P **is** B_{n+1} **endexit**" part is optional, specifies trapping and handling of various events in B. Each X_i, possibly carrying data IPL_i, is a signal trapped as an exception and transferring control from B to B_i, while "**exit** P **is** B_{n+1} **endexit**", where data P is optional, specifies that δ in B transfers control to B_{n+1}. A shorthand for the case where only δ is trapped is "$B; B_{n+1}$", i.e. sequential composition of B and B_{n+1}, where all data generated by B is available to B_{n+1}. "**loop** B **endloop**" denotes an infinite sequence of processes B.

A "$B_1[]B_2$" denotes a process behaving as B_1 or as B_2, where the choice is made upon the first event. Alternatives with an initial δ are not foreseen.

A "$B_1[X > B_2$" denotes process B_1 repeatedly suspended upon the start of B_2. When B_1 becomes ready for a δ, the composite process may execute an auxiliary **i** leading to successful termination. For B_2, it is expected that it has neither an initial δ nor an initial signal X. Whenever X occurs in B_2, it is implicitly trapped as an exception, and consequently, B_1 is resumed and B_2 reset to its initial state. A shorthand for the case where B_1 is never

$$\frac{B_k \xrightarrow{\mathbf{i}} B'_k}{\begin{array}{l}\mathbf{par}\ O_1\#N_1,\ldots,O_p\#N_p\ \mathbf{in}\ [\Omega_1]\to B_1\ ||\ldots||\ [\Omega_m]\to B_m\ \mathbf{endpar}\\ \xrightarrow{\mathbf{i}} \mathbf{par}\ O_1\#N_1,\ldots,O_p\#N_p\ \mathbf{in}\ [\Omega_1]\to B'_1\ ||\ldots||\ [\Omega_m]\to B'_m\ \mathbf{endpar}\end{array}} \begin{bmatrix} k\in\{1,\ldots,m\} \\ \forall i\neq k: B'_i = B_i\end{bmatrix} \text{(PAR1)}$$

$$\frac{B_k \xrightarrow{X(RN)} B'_k}{\begin{array}{l}\mathbf{par}\ O_1\#N_1,\ldots,O_p\#N_p\ \mathbf{in}\ [\Omega_1]\to B_1\ ||\ldots||\ [\Omega_m]\to B_m\ \mathbf{endpar}\\ \xrightarrow{\mathbf{i}} \mathbf{signal}\ X(RN); \mathbf{par}\ O_1\#N_1,\ldots,O_p\#N_p\ \mathbf{in}\ [\Omega_1]\to B'_1\ ||\ldots||\ [\Omega_m]\to B'_m\ \mathbf{endpar}\end{array}} \begin{bmatrix} k\in\{1,\ldots,m\} \\ \forall i\neq k: B'_i = B_i\end{bmatrix} \text{(PAR2)}$$

$$\frac{\begin{array}{c}\forall i\in\Sigma: B_i \xrightarrow{O(RN)} B'_i, \forall i\in\Sigma': B_i \xrightarrow{R_O(RN)} B'_i,\\ \forall i\in(\{1,\ldots,m\}\backslash(\Sigma+\Sigma')): B_i \not\xrightarrow{R_O(RN)}\end{array}}{\begin{array}{l}\mathbf{par}\ O_1\#N_1,\ldots,O_p\#N_p\ \mathbf{in}\ [\Omega_1]\to B_1\ ||\ldots||\ [\Omega_m]\to B_m\ \mathbf{endpar}\\ \xrightarrow{O(RN)} \mathbf{par}\ O_1\#N_1,\ldots,O_p\#N_p\ \mathbf{in}\ [\Omega_1]\to B'_1\ ||\ldots||\ [\Omega_m]\to B'_m\ \mathbf{endpar}\end{array}} \begin{bmatrix} Exec(O,\Sigma) \\ \Sigma'\subseteq(\{1,\ldots,m\}\backslash\Sigma) \\ \forall i\notin(\Sigma+\Sigma'): B'_i = B_i\end{bmatrix} \text{(PAR3)}$$

$$\begin{array}{l}\underline{\text{where}}\ Exec(O,\Sigma) := ((\emptyset\subset\Sigma\subseteq\{1,\ldots,m\})\wedge\\ \quad((\exists i: ((\Sigma=\{i\})\wedge(O\notin\Omega_i)))\vee\\ \quad((\forall i\in\Sigma: (O\in\Omega_i))\wedge\\ \quad((\exists k\in\{1,\ldots,p\}: ((O=O_k)\wedge(|\Sigma|=N_k)))\vee\\ \quad((\forall k\in\{1,\ldots,p\}: (O\neq O_k))\wedge(\forall i\in(\{1,\ldots,m\}\backslash\Sigma): (O\notin\Omega_i)))))))\end{array}$$

$$\frac{\forall i\in\Sigma': B_i \xrightarrow{R_O(RN)} B'_i, \forall i\in(\{1,\ldots,m\}\backslash\Sigma'): B_i \not\xrightarrow{R_O(RN)}}{\begin{array}{l}\mathbf{par}\ O_1\#N_1,\ldots,O_p\#N_p\ \mathbf{in}\ [\Omega_1]\to B_1\ ||\ldots||\ [\Omega_m]\to B_m\ \mathbf{endpar}\\ \xrightarrow{R_O(RN)} \mathbf{par}\ O_1\#N_1,\ldots,O_p\#N_p\ \mathbf{in}\ [\Omega_1]\to B'_1\ ||\ldots||\ [\Omega_m]\to B'_m\ \mathbf{endpar}\end{array}} \begin{bmatrix} \emptyset\subset\Sigma'\subseteq\{1,\ldots,m\} \\ \forall i\notin\Sigma': B'_i = B_i\end{bmatrix} \text{(PAR4)}$$

$$\frac{\forall i\in\{1,\ldots,m\}: B_i \xrightarrow{\delta(RN_i)} B'_i}{\begin{array}{l}\mathbf{par}\ O_1\#N_1,\ldots,O_p\#N_p\ \mathbf{in}\ [\Omega_1]\to B_1\ ||\ldots||\ [\Omega_m]\to B_m\ \mathbf{endpar}\\ \xrightarrow{\delta(RN_1,\ldots,RN_m)} \mathbf{par}\ O_1\#N_1,\ldots,O_p\#N_p\ \mathbf{in}\ [\Omega_1]\to B'_1\ ||\ldots||\ [\Omega_m]\to B'_m\ \mathbf{endpar}\end{array}} \text{(PAR5)}$$

Figure 1: Untimed dynamic semantics of parallel composition enhanced with broadcasting

resumed is "$B_1[> B_2$".

A "**par** $G_1\#N_1,\ldots,G_p\#N_p$ **in** $[\Gamma_1]\to B_1||\ldots||[\Gamma_m]\to B_m$ **endpar**" denotes parallel composition of processes B_1 to B_m. Each B_i is associated with a Γ_i listing the gates on which B_i synchronizes with its peers. If the gate G on which a synchronization occurs has its synchronization degree N defined in the list "$G_1\#N_1,\ldots,G_p\#N_p$", that is a synchronization of exactly N processes B_i with G in Γ_i, otherwise it is a synchronization of all such processes. The composite process successfully terminates when all its constituents do. A shorthand for processes B_1 and B_2 synchronized on gates $G_1,\ldots,G_n$ is "$B_1|[G_1,\ldots,G_n]|B_2$". Shorthands for the minimum and the maximum synchronization are "$B_1|||B_2$" and "$B_1||B_2$", respectively.

A "**rename gate** $G_1(IPL_1)$ **is** $G'_1\ P_1\ \ldots$ **gate** $G_m(IPL_m)$ **is** $G'_m\ P_m$ **signal** $X_1(IPL'_1)$ **is** $X'_1\ E_1\ldots$ **signal** $X_n(IPL'_n)$ **is** $X'_n\ E_n$ **in** B **endren**" denotes process B with some actions and signals renamed.

A "**hide** $G_1 : T_1,\ldots,G_n : T_n$ **in** B **endhide**" denotes process B with all its actions on gates G_1 to G_n of the respective types T_1 to T_n hidden.

A "**var** $V_1 : T_1 := E_1,\ldots,V_n : T_n := E_n$ **in** B **endvar**" declares for process B some variables V_i, respectively of type T_i and optionally initialized to E_i. A "$?V := E$" sets variable V to E.

Where we use "[]" or "|||" as a prefix operator, composition of an empty set of processes denotes a **stop**.

3 Parallel Composition Enhanced with Broadcasting

We define that $\mathcal{G}$, the universe of gates, consists of disjoint parts $\mathcal{O}$ and $\mathcal{R}$. $\mathcal{O}$ is the universe of gates O, of type T_O, for so called ordinary actions. For the purpose of broadcasting, gates in $\mathcal{O}$ are interpreted as "transmission" gates. In $\mathcal{R}$, there is a unique "reception" gate R_O, of type T_{R_O}, for every O in $\mathcal{O}$. We assume that T_{R_O} and T_O denote the same type.

Fig. 1 defines the desired enhancements of the parallel composition operator.

PAR1: Every internal action is executed individually by a B_k.

PAR2: Every signal is issued individually by a B_k.

PAR3: Every ordinary action $O(RN)$ involves not only its executors (their indexes are listed in Σ, where $Exec(O, \Sigma)$ enforces the usual E-LOTOS rules for process synchronization), but also its passive observers, i.e. processes ready to execute a "reception" $R_O(RN)$ of the data RN "transmitted" by the action $O(RN)$. Note that RN is also "transmitted" into the environment of the composite process.

PAR4: Whenever an RN is "transmitted" by an $O(RN)$ executed in the environment of the composite process, it is "received" by those among processes B_1 to B_m which are ready for an $R_O(RN)$.

PAR5: B_1 to B_m can successfully terminate only in co-operation.

Note that in an $O(RN)$, RN says nothing about the direction in which the data flows in the event. One may, for example, as well synchronize an "$O?V$" and an "$R_O!E$", i.e. let the data flow from a "recipient" to a "transmitter". The only concept we newly introduce is the concept of passive observers. The concept has a wide applicability, where the classical broadcast is just its special case and a matter of specification style.

4 Specifying Broadcast Communication in the Original E-LOTOS

4.1 Notation

As first, we extend $\mathcal{G}$ with some auxiliary gates. For every O in $\mathcal{O}$, we introduce three different new gates of type T_O: I_O, A_O and A_{R_O}. Let $\mathcal{I}$ denote the universe of gates I_O. We will also need a special gate G_δ.

In the following definitions, the behaviour of a B is assumed to be the one defined by the enhanced process semantics.

$\mathcal{G}(B)$ lists the visible gates of B, $\mathcal{O}(B)$ its visible gates from $\mathcal{O}$, and $\mathcal{R}(B)$ all the O with R_O in $\mathcal{G}(B)$. $\mathcal{X}(B)$ lists the exceptions raised but not trapped by B. $VarOK(B)$ indicates whether subprocesses of B access data variables in a consistent way. $Term(B)$ indicates whether B might successfully terminate.

$\mathcal{S}(B)$ is a set of gates. If it is non-empty, this means that all starting events of B are observable actions, where $\mathcal{S}(B)$ lists the gates on which they occur.

$\mathcal{L}(B)$ is a set of gates. If it is non-empty, this means that any action of B on one of the gates immediately (i.e. over a finite number of actions which are all immediate and internal to B) leads to successful termination, and that δ in B is always preceded by such an action, so that the environment of B can detect its successful termination by synchronizing on the gates in $\mathcal{L}(B)$, i.e. on its last visible action before δ.

$\mathcal{L}_X(B)$ is a set of gates. If it is non-empty, this means that any action of B on one of the gates immediately (i.e. over a finite number of actions which are all immediate and internal to B) leads to exception X and that raising of X in B is always preceded by such an action, so that the environment of B can detect its exception X by synchronizing on the gates in $\mathcal{L}_X(B)$, i.e. on its last visible action before X.

4.2 Principles of Implementation

In this section, we define a transformation $M_\rho(B)$ which takes an E-LOTOS process B defined on gates in $(\mathcal{O} \cup \mathcal{R})$ and generates a process implementing its new semantics. The parameter ρ lists those gates O on which the environment "transmits" to B. If B is not further combined, ρ is empty.

The main problem to solve is implementation of negative premises in the enhanced semantics of the parallel composition operator. As we see in Fig. 1, a process B_i with $i \notin (\Sigma \cup \Sigma')$ doesn't participate in an $O(RN)$ because it doesn't have $R_O(RN)$ enabled. A positivistic re-interpretation says that B_i *does* participate in the event – by ignoring it, i.e. by executing an $I_O(RN)$ without changing its state.

An $M_\rho(B)$ has to be active on a gate I_O only if $O \in \rho$, and (according to our implementation strategy) only as long as it does not enter a state where δ or an exception signal is its only possible next step.

Another crucial concept we use is gate splitting. It requires [1] that every gate action carries an auxiliary parameter of type "nat", i.e. ranging over all natural numbers, including 0.

4.3 Implementation of Simple Processes and Operators

The simple cases of M_ρ are defined in Fig. 2.

$M_\rho(\mathbf{stop})$ is just permanently ready to ignore actions on gates in ρ.

$M_\rho(\mathbf{null})$ is simply "**null**", because we assume that

$$M_\rho(\mathbf{stop}) := (|||_{O\in\rho}\mathbf{loop}\ I_O(\mathbf{any} : \mathrm{nat}, \mathbf{any} : T_O)\ \mathbf{endloop}) \qquad M_\rho(\mathbf{null}) := \mathbf{null}$$
$$M_\rho(O\ P_1@P_2[E]) := (M_\rho(\mathbf{stop})[> O(\mathbf{any} : \mathrm{nat}, P_1)@P_2[E]) \qquad M_\rho(\mathbf{i}) := (M_\rho(\mathbf{stop})[> \mathbf{i})$$
$$M_\rho(\mathbf{signal}\ X(E)) := (M_\rho(\mathbf{stop})[> \mathbf{signal}\ X(E)) \qquad M_\rho(\mathbf{raise}\ X(E)) := \mathbf{raise}\ X(E)$$
$$M_\rho(R_O\ P_1@P_2[E]) := ((M_{\rho\setminus\{O\}}(\mathbf{stop})$$
$$\underline{\text{if}}\ O \in \rho\ \underline{\text{then}}\ |||\mathbf{var}\ x : T_O, t : \mathrm{time}, t' : \mathrm{time} := 0\ \mathbf{in}$$
$$\mathbf{loop}\ I_O(\mathbf{any} : \mathrm{nat}, ?x)@t[E'(t'+t)]; ?t' := t'+t\ \mathbf{endloop}\ \mathbf{endvar}\ \underline{\text{endthen}})$$
$$[> R_O(\mathbf{any} : \mathrm{nat}, P_1)@P_2[E])$$
$$\underline{\text{where}}\ (((R_O\ P_1@P_2[E][]\mathbf{var}\ x : T_O, t : \mathrm{time}\ \mathbf{in}\ R_O?x@t[E'(t)]\ \mathbf{endvar}) \sim R_O\ \mathbf{any} : T_O)$$
$$\wedge((R_O\ P_1@P_2[E]|||\mathbf{var}\ x : T_O, t : \mathrm{time}\ \mathbf{in}\ R_O?x@t[E'(t)]\ \mathbf{endvar}) \sim \mathbf{stop}))$$
$$M_\rho(\mathbf{trap\ exception}\ X_1(IPL_1)\ \mathbf{is}\ B_1\ \mathbf{endexn}\ \ldots\ \mathbf{exit}\ P\ \mathbf{is}\ B_{n+1}\ \mathbf{endexit\ in}\ B\ \mathbf{endtrap})$$
$$:= \mathbf{trap\ exception}\ X_1(IPL_1)\ \mathbf{is}\ M_\rho(B_1)\ \mathbf{endexn}\ \ldots\ \mathbf{exit}\ P\ \mathbf{is}\ M_\rho(B_{n+1})\ \mathbf{endexit\ in}\ M_\rho(B)\ \mathbf{endtrap}$$
$$M_\rho(\mathbf{hide}\ O_1 : T_1, \ldots, O_n : T_n\ \mathbf{in}\ B\ \mathbf{endhide} := \mathbf{hide}\ O_1 : T_1, \ldots, O_n : T_n\ \mathbf{in}\ M_\rho(B)\ \mathbf{endhide}$$
$$M_\rho(\mathbf{rename\ gate}\ O_1(IPL_1)\ \mathbf{is}\ O_1'\ P_1\ \ldots\ \mathbf{signal}\ X_1(IPL_1')\ \mathbf{is}\ X_1'\ E_1\ \ldots\ \mathbf{in}\ B\ \mathbf{endren})$$
$$:= \mathbf{rename\ gate}\ O_1(IPL_1)\ \mathbf{is}\ O_1'\ P_1\ \ldots\ \mathbf{signal}\ X_1(IPL_1')\ \mathbf{is}\ X_1'\ E_1\ \ldots\ \mathbf{in}\ M_\rho(B)\ \mathbf{endren}$$

Figure 2: Implementation of some simple processes and operators

$$M_\rho(B_1[]B_2\ \underline{\text{where}}\ ((\rho \neq \emptyset) \Rightarrow (VarOK(B_1|||B_2) \wedge (\forall i \in \{1,2\} : ((\mathcal{S}(B_i) \neq \emptyset) \wedge (Term(B_i) \Rightarrow (\mathcal{L}(B_i) \neq \emptyset)))))))$$
$$:= \underline{\text{if}}\ \rho = \emptyset\ \underline{\text{then}}\ (M_\rho(B_1)[]M_\rho(B_2))$$
$$\underline{\text{else}}\ \mathbf{rename}\ \underline{\text{forall}}\ G \in Sync : \mathbf{gate}\ G(?x : \mathrm{nat}, ?y : T_G, \mathbf{etc})\ \mathbf{is}\ G(!x, !y)\ \underline{\text{endfor}}$$
$$\mathbf{in}\ C_1|[Sync \cup \{I_O|(O \in \rho)\}]|C_2\ \mathbf{endren}$$
$$\underline{\text{where}}\ Sync := (\mathcal{S}(B_1) \cup \mathcal{L}(B_1) \cup \mathcal{S}(B_2) \cup \mathcal{L}(B_2))$$
$$C_i := ((\mathbf{rename}\ \underline{\text{forall}}\ G \in (\mathcal{G}(B_i) \cap Sync) : \mathbf{gate}\ G(?x : \mathrm{nat}, ?y : T_G)\ \mathbf{is}\ G(!x, !y, !i)\ \underline{\text{endfor}}$$
$$\mathbf{in}\ M_\rho(B_i)\ \mathbf{endren}$$
$$[> (([]_{G\in(\mathcal{S}(B_{i'})\setminus\mathcal{L}(B_{i'}))}G(\mathbf{any} : \mathrm{nat}, \mathbf{any} : T_G, !i'));$$
$$((|||_{G\in((\mathcal{G}(B_{i'})\cap Sync)\setminus\mathcal{L}(B_{i'}))}\mathbf{loop}\ G(\mathbf{any} : \mathrm{nat}, \mathbf{any} : T_G, !i')\ \mathbf{endloop})|||M_\rho(\mathbf{stop}))))$$
$$[> ([]_{G\in\mathcal{L}(B_{i'})}G(\mathbf{any} : \mathrm{nat}, \mathbf{any} : T_G, !i')))\ \underline{\text{where}}\ (\{i, i'\} = \{1,2\})\ \underline{\text{endelse}}$$

Figure 3: Implementation of choice

null is just a placeholder for the handler of the δ it represents. Likewise, a "**raise** $X(E)$" is a placeholder for the handler of $X(E)$, hence unaffected by M_ρ.

An $M_\rho(\mathbf{i})$, an $M_\rho(O\ P_1@P_2[E])$ with P_1 of type T_O, or an $M_\rho(\mathbf{signal}\ X(E))$ continuously ignores all external actions on gates in ρ, until it executes the specified **i**, O or X, respectively, and thereby reduces to successful termination.

Likewise, an $M_\rho(R_O\ P_1@P_2[E])$ executes actions on gates in $\mathcal{I}$ until it executes the specified "reception" and thereby reduces to successful termination. It is, however, rather complicated to specify execution of I_O actions. For particular data of type T_O, they must be enabled exactly when "reception" of such data is not. One must find a suitable predicate E' for their guarding. In a general case, the predicate must be a function of the time elapsed from the moment when the process $M_\rho(R_O\ P_1@P_2[E])$ was enabled. As an additional complication, the loop can directly measure only the time t elapsed from the last I_O action, where auxiliary variable t' remembers the moment of its occurrence.

Transformation M_ρ commutes with the operator of trapping. It also commutes with hiding, because we restrict it to hiding of ordinary actions, and with renaming, because we restrict it to renaming within the universes of ordinary actions and signals.

4.4 Implementation of Choice

For a B specified as "$B_1[]B_2$", $M_\rho(B)$ is defined in Fig. 3. It combines $M_\rho(B_1)$ and $M_\rho(B_2)$, because ignoring of an external "transmission" requires that both parts do it.

If ρ is empty, the first event in $M_\rho(B_1)$ or $M_\rho(B_2)$ is not on a gate in $\mathcal{I}$, hence it may resolve the choice. This means that the two parts may be combined by the operator of choice.

If ρ is non-empty, the first event might be on a gate in $\mathcal{I}$, i.e. must not resolve the choice. So $M_\rho(B)$ is specified in the constraint-oriented style [4], with a constraint C_i for each alternative B_i. For each alternative, we require that its start and its successful termination (if any) are detectable through synchronization. As C_1 and C_2 are concurrent, $VarOK(B_1|||B_2)$ is required.

$$
\begin{array}{l}
M_\rho(B_1[X > B_2 \ \underline{\text{where}}\ ((\rho \neq \emptyset) \Rightarrow (VarOK(B_1|||B_2) \wedge (\mathcal{S}(B_2) \neq \emptyset) \wedge (Term(B_2) \Rightarrow (\mathcal{L}(B_2) \neq \emptyset)) \wedge \\
\qquad\qquad\qquad\qquad ((X \in \mathcal{X}(B_2)) \Rightarrow (\mathcal{L}_X(B_2) \neq \emptyset)))))) \\
:= \underline{\text{if}}\ \rho = \emptyset\ \underline{\text{then}}\ (M_\rho(B_1)[X > M_\rho(B_2)) \\
\quad \underline{\text{else}}\ \mathbf{rename}\ \underline{\text{forall}}\ G \in (\mathcal{G}(B_1) \cap Sync) : \mathbf{gate}\ A_G\ \mathbf{is}\ G\ \underline{\text{endfor}} \\
\qquad \mathbf{in\ hide}\ G_\delta\ \mathbf{in}\ C_1|[Sync \cup \{G_\delta\} \cup \{I_O|(O \in \rho)\}]|C_2\ \mathbf{endhide\ endren} \\
\qquad \underline{\text{where}}\ Sync := (\mathcal{S}(B_2) \cup \mathcal{L}(B_2) \cup \mathcal{L}_X(B_2)) \\
\qquad\quad C_1 := (((\mathbf{rename}\ \underline{\text{forall}}\ G \in (\mathcal{G}(B_1) \cap Sync) : \mathbf{gate}\ G\ \mathbf{is}\ A_G\ \underline{\text{endfor}}\ \mathbf{in}\ M_\rho(B_1)\ \mathbf{endren}; \\
\qquad\qquad\qquad (M_\rho(\mathbf{stop})[> G_\delta)) \\
\qquad\qquad\qquad [X > ((([]_{G \in (\mathcal{S}(B_2) \setminus (\mathcal{L}(B_2) \cup \mathcal{L}_X(B_2)))} G(\mathbf{any} : \mathrm{nat}, \mathbf{any} : T_G)); \\
\qquad\qquad\qquad\qquad ((|||_{G \in (\mathcal{S}(B_2) \setminus (\mathcal{L}(B_2) \cup \mathcal{L}_X(B_2)))} \mathbf{loop}\ G(\mathbf{any} : \mathrm{nat}, \mathbf{any} : T_G)\ \mathbf{endloop})|||M_\rho(\mathbf{stop}))) \\
\qquad\qquad\qquad\quad [> (([]_{G \in \mathcal{L}_X(B_2)} G(\mathbf{any} : \mathrm{nat}, \mathbf{any} : T_G)); \mathbf{raise}\ X))) \\
\qquad\qquad\qquad [> ([]_{G \in \mathcal{L}(B_2)} G(\mathbf{any} : \mathrm{nat}, \mathbf{any} : T_G))) \\
\qquad\quad C_2 := ((\mathbf{stop}[X > M_\rho(B_2))\ \underline{\text{if}}\ Term(B_1)\ \underline{\text{then}}\ [> G_\delta\ \underline{\text{endthen}})\ \underline{\text{endelse}}
\end{array}
$$

Figure 4: Implementation of suspend/resume

Where necessary, visible actions of B are internally to $M_\rho(B)$ furnished with the index i of the alternative B_i to which they belong. Each C_i states the following, where $B_{i'}$ denotes the other alternative:

1. Execute $M_\rho(B_i)$ until $M_\rho(B_{i'})$ executes an action on a gate in $\mathcal{S}(B_{i'})$. If such disabling of $M_\rho(B_i)$ occurs, continue supporting execution of $M_\rho(B_{i'})$.
2. When $M_\rho(B_i)$ successfully terminates or when $M_\rho(B_{i'})$ executes an action on a gate in $\mathcal{L}(B_{i'})$, successfully terminate.

4.5 Implementation of Suspend/Resume

For a B specified as "$B_1[X > B_2$", $M_\rho(B)$ is defined in Fig. 4. It combines $M_\rho(B_1)$ and $M_\rho(B_2)$, because ignoring of an external "transmission" requires that both parts do it. A hidden G_δ implements the implicitly specified **i** introducing successful termination of B upon successful termination of B_1.

If ρ is empty, the first event in $M_\rho(B_2)$ is not on a gate in $\mathcal{I}$, hence it may disable $M_\rho(B_1)$. This means that the two parts may be combined by the suspend/resume operator.

If ρ is non-empty, the first event in $M_\rho(B_2)$ might be on a gate in $\mathcal{I}$, i.e. must not disable $M_\rho(B_1)$. So $M_\rho(B)$ is specified in the constraint-oriented style, with a constraint C_i for each B_i. For B_2, we require that its start, its successful termination (if any) and its exception X (if any) are detectable through synchronization. $VarOK(B_1|||B_2)$ is required.

Where necessary, gates G of B_1 are internally to $M_\rho(B)$ renamed into A_G, to differ from gates G of B_2. C_1 states the following:

1. Execute $M_\rho(B_1)$ until $M_\rho(B_2)$ executes an action on a gate in $\mathcal{S}(B_2)$. If such suspension of $M_\rho(B_1)$ occurs, continue supporting execution of $M_\rho(B_2)$.
2. When $M_\rho(B_2)$ executes an action on a gate in $\mathcal{L}_X(B_2)$, resume $M_\rho(B_1)$.
3. After $M_\rho(B_1)$ becomes ready for successful termination, try to execute a G_δ and successfully terminate. In the meantime, execute $M_\rho(\mathbf{stop})$.
4. When $M_\rho(B_2)$ executes an action on a gate in $\mathcal{L}(B_2)$, successfully terminate.

C_2 states the following:

1. Execute $M_\rho(B_2)$, restarting it upon exception X.
2. When $M_\rho(B_2)$ successfully terminates or when G_δ occurs, successfully terminate.

4.6 Implementation of Parallel Composition

For a B specified as parallel composition of processes B_i, $M_\rho(B)$ is defined in Fig. 5. It combines processes $M_{\mathcal{I}(B_i)}(B_i)$, where $\mathcal{I}(B_i)$ is ρ extended with O on which the peers of B_i "transmit" or "receive".

Processes $M_{\mathcal{I}(B_i)}(B_i)$ are fully synchronized, i.e. every O, R_O or I_O action requires co-operation of all the processes. In such an action, each individual $M_{\mathcal{I}(B_i)}(B_i)$ participates as a "transmitter" (by executing an O action), as a "receiver" (by executing an R_O action), or as an "ignorer" (by executing an I_O action). Like in Fig. 1, let Σ identify the "transmitters" and Σ' the "receivers". Because for an O, R_O or I_O there might be different possible combinations of participants' roles, actions internally to

$$
\begin{array}{l}
M_\rho(\mathbf{par}\ O_1\#N_1,\ldots,O_p\#N_p\ \mathbf{in}\ [\Omega_1]\rightarrow B_1 \ ||\ldots||\ [\Omega_m]\rightarrow B_m\ \mathbf{endpar} \\
\quad \underline{\text{where}}\ \forall i\in\{1,\ldots,m\}:(((\mathcal{I}(B_i)\neq\emptyset)\wedge Term(B_i))\Rightarrow((\mathcal{L}(B_i)\neq\emptyset)\wedge(\mathcal{L}(B_i)=\mathcal{L}(B)))) \\
:= \mathbf{rename}\ \underline{\text{forall}}\ O\in(\cup_{i=1\ldots m}\mathcal{O}(B_i)):\mathbf{gate}\ O(?x:\text{nat},?y:T_O,\mathbf{etc})\ \mathbf{is}\ O(!x,!y)\ \underline{\text{endfor}} \\
\qquad\qquad \underline{\text{forall}}\ O\in(\cup_{i=1\ldots m}\mathcal{R}(B_i)):\mathbf{gate}\ R_O(?x:\text{nat},?y:T_O,\mathbf{etc})\ \mathbf{is}\ R_O(!x,!y)\ \underline{\text{endfor}} \\
\qquad\qquad \underline{\text{forall}}\ O\in\rho:\mathbf{gate}\ I_O(?x:\text{nat},?y:T_O,\mathbf{etc})\ \mathbf{is}\ I_O(!x,!y)\ \underline{\text{endfor}} \\
\quad \mathbf{in}\ ||_{i=1\ldots m}C_i\ \mathbf{endren} \\
\quad \underline{\text{where}}\ C_i := \mathbf{rename}\ \underline{\text{forall}}\ O\in\mathcal{O}(B_i):Split(O,T_O,Act(O,i),i)\ \underline{\text{endfor}} \\
\qquad\qquad\qquad\qquad \underline{\text{forall}}\ O\in\mathcal{R}(B_i):Split(R_O,T_O,Act(R_O,i),i)\ \underline{\text{endfor}} \\
\qquad\qquad\qquad\qquad \underline{\text{forall}}\ O\in\mathcal{I}(B_i):Split(I_O,T_O,Act(I_O,i),i)\ \underline{\text{endfor}} \\
\qquad\qquad \mathbf{in}\ M_{\mathcal{I}(B_i)}(B_i)\ \mathbf{endren} \\
\qquad\qquad \underline{\text{where}}\ \mathcal{I}(B_i) := (\rho\cup\{O|\exists\Sigma\subseteq(\{1,\ldots,m\}\backslash\{i\}):(Exec(O,\Sigma)\wedge\forall k\in\Sigma:(O\in\mathcal{O}(B_k)))\} \\
\qquad\qquad\qquad\qquad \cup(\cup_{j\in(\{1,\ldots,m\}\backslash\{i\})}\mathcal{R}(B_j))) \\
\qquad\qquad\quad Act(G,i) := (\{O(\Sigma,\Sigma')|(Exec(O,\Sigma)\wedge(\Sigma'\subseteq(\{1,\ldots,m\}\backslash\Sigma))\wedge \\
\qquad\qquad\qquad\qquad\qquad (\forall k\in\Sigma:(O\in\mathcal{O}(B_k)))\wedge(\forall k\in\Sigma':(R_O\in\mathcal{O}(B_k)))\wedge \\
\qquad\qquad\qquad\qquad\qquad (((G=O)\wedge(i\in\Sigma))\vee((G=R_O)\wedge(i\in\Sigma'))\vee \\
\qquad\qquad\qquad\qquad\qquad ((G=I_O)\wedge(i\notin(\Sigma+\Sigma'))))))\}+ \\
\qquad\qquad\qquad\quad \{R_O(\{\},\Sigma')|((\emptyset\subset\Sigma'\subseteq\{1,\ldots,m\})\wedge(\forall k\in\Sigma':(R_O\in\mathcal{O}(B_k)))\wedge \\
\qquad\qquad\qquad\qquad\qquad (((G=R_O)\wedge(i\in\Sigma'))\vee((G=I_O)\wedge(i\notin\Sigma'))))\}+ \\
\qquad\qquad\qquad\quad \{I_O(\{\},\{\})|((O\in\rho)\wedge(G=I_O))\}) \\
\qquad\qquad\quad Split(G,T,\emptyset,i) := \mathbf{gate}\ G(\mathbf{etc})\ \mathbf{is}\ G!i \\
\qquad\qquad\quad Split(G,T,\{G_0(\Sigma_0,\Sigma'_0),\ldots,G_{n-1}(\Sigma_{n-1},\Sigma'_{n-1})\},i) := \\
\qquad\qquad\qquad \underline{\text{forall}}\ j\in\{0,\ldots,n-1\}:\mathbf{gate}\ G(?x:\text{mod\_}j\text{\_}n,?y:T)\ \mathbf{is}\ G_j(!(x\ \text{div}\ n),!y,!\Sigma_j,!\Sigma'_j)\ \underline{\text{endfor}}
\end{array}
$$

Figure 5: Implementation of parallel composition

$M_\rho(B)$ carry Σ and Σ' as additional parameters.

For a G in a $\mathcal{G}(M_{\mathcal{I}(B_i)}(B_i))$, $Act(G,i)$ lists the $O(\Sigma,\Sigma')$, $R_O(\Sigma,\Sigma')$ and $I_O(\Sigma,\Sigma')$ into which it has to be split, because it might have to synchronize on them. The renamings necessary for such splitting are generated by function $Split$.

$Split$ splits an action with respect to its first parameter. The parameter x ranges over all natural numbers. If an action is split into n variants, the j-th variant gets all x which are of a type "mod_$(j-1)$_n", i.e. with $(x \bmod n = j-1)$. The first parameter of the generated variants ranges over $(x\ \text{div}\ n)$, hence also over all natural numbers, as required. In the exceptional case where an $Act(G,i)$ is empty, G actions of $M_{\mathcal{I}(B_i)}(B_i)$ are renamed into $G!i$, so that they become non-executable within the context of $M_\rho(B)$, for there is no action in which $M_{\mathcal{I}(B_i)}(B_i)$ should participate by a G.

If an $M_{\mathcal{I}(B_i)}(B_i)$ becomes ready to successfully terminate before an $M_{\mathcal{I}(B_{i'})}(B_{i'})$ does, it might block further activities of the composite process. So we require that the concurrent processes always become ready for successful termination in a co-ordinated manner, by synchronizing on a G in $\mathcal{L}(B)$.

5 Conclusion

We have generalized (though with some restrictions) the method of [1] to systems comprising timed actions, process suspension and resumption, exception raising and handling, and "m-among-n" synchronization of processes. The proposed implementation transformations might seem complicated, but as they can be easily mechanized, this is not a problem.

References

[1] M. Kapus-Kolar, Specifying Broadcast Communication in a Sublanguage of E-LOTOS, in *EUROCON 2003 - Computer as a Tool*, IEEE Computer Society Press, 2003, vol. II, pp. 2–6.

[2] ISO/IEC, *Enhancements to LOTOS (E-LOTOS)*, ISO/IEC 15437, ISO – Information Technology, 2001.

[3] A. Verdejo, *E-LOTOS: Tutorial and Semantics*, M.S. thesis, Universidad Complutense de Madrid, 1999.

[4] C. A. Vissers, G. Scollo, M. van Sinderen, and H. Brinksma, Specification Styles in Distributed Systems Design and Verification, *Theoretical Computer Science*, vol. 89, 1991, 179–206.