Elektrotehniški vestnik 71(5): 1–7, 2004
Electrotechnical Review, Ljubljana, Slovenija

Specifying Late Choice in E-LOTOS

Monika Kapus-Kolar

Jožef Stefan Institute, Dept. of Digital Communications and Networks, Jamova 39, 1000 Ljubljana, Slovenia
E-mail: monika.kapus-kolar@ijs.si

Abstract. For a pair of alternative processes, late choice means that they proceed concurrently in co-operation until one of them executes an action on which the other is not able or not allowed to synchronize. In incremental system specification, late choice is the key concept. We show how it can be specified in E-LOTOS, a standard language for specification of concurrent and reactive systems. The proposed solutions are helpful also in specification of dependence-based action ordering. The discussion is limited to a Basic-LOTOS-like sublanguage of E-LOTOS.

Key words: concurrent systems, formal specification, E-LOTOS, late choice, dependence-based action ordering

Specifikacija poznega izbora v jeziku E-LOTOS

Povzetek. Za par alternativnih procesov pozen izbor pomeni, da tečeta vzporedno in sodelujeta, dokler kateri od njiju ne izvede akcije, na katero se drugi ne more ali ne sme sinhronizirati. Pri inkrementalni specikaciji sistemov je pozen izbor ključni koncept. V članku pokažemo, kako ga je mogoče opisati v jeziku E-LOTOS, standardnem jeziku za specifikacijo vzporednih in reaktivnih sistemov. Predlagane rešitve pomagajo tudi pri specifikaciji vrstnega reda akcij glede na njihovo medsebojno odvisnost. V razpravi se omejimo na podjezik jezika E-LOTOS, ki je podoben osnovnemu delu jezika LOTOS.

Ključne besede: vzporedni sistemi, formalna specifikacija, E-LOTOS, pozen izbor, odvisnostno urejanje akcij

1 Introduction

When a user communicates with a server, she/he usually expects that its behaviour is deterministic. In other words, whenever the server chooses between a pair of possible scenarios on its interface, it is supposed to delay the choice until the scenarios begin to differ. So when specifying such a system by gradually extending the set of its execution scenarios, it is convenient if the employed specification language contains an operator of late choice.

LOTOS [1], a standard process algebraic language for specification of concurrent and reactive systems, contains no such operator, although one has been proposed by Ichikawa & al. [3]. In E-LOTOS [2, 8], the enhanced successor of LOTOS, there is also no such operator, but below we show that late decisions can in many cases be specified indirectly. The discussion is limited to a Basic-LOTOS-like sublanguage of E-LOTOS.

The paper is organized as follows. Sect. 2 is a brief introduction to the employed sublanguage of E-LOTOS. Sect. 3 defines the class of processes for which we would like to implement delayed decisions, and the desired enhanced operational semantics. Sect. 4 describes a transformation for encoding the enhancements in the original E-LOTOS.

Received 8 March 2004
Accepted 14 December 2004

2 Specification Language

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

"**stop**" denotes inaction.

"**null**" denotes successful termination.

A "G" specifies an observable action, i.e. an interaction of the specified process at gate G. Interactions may be associated with data, and gates may be typed with the type of the values they can communicate.

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.

A "$B_1 ; B_2$" denotes sequential composition, i.e. B_1 upon its successful termination followed by B_2. A shorthand for an infinite sequence of processes B is "**loop** B **endloop**".

A "$B_1 |[G_1, \ldots, G_n]| B_2$" denotes processes B_1 and B_2 running in parallel and synchronized on gates $G_1, \ldots, G_n$. The usual shorthands for pure interleaving and full synchronization are "$B_1 ||| B_2$" and "$B_1 || B_2$", respectively.

A "$B_1 [> B_2$" denotes process B_1 potentially disabled upon the start of B_2.

A "**rename gate** G_1 **is** $G'_1 \ldots$ **gate** G_n **is** G'_n **in** B **endren**" denotes process B with some of its actions renamed as specified. More selective renaming can be achieved through gate typing. With this operator, one

$$\mathbf{null} \xrightarrow{\delta} \mathbf{stop} \quad O \xrightarrow{O} \mathbf{null} \quad \mathbf{loop}\ O\ \mathbf{endloop} \xrightarrow{O} \mathbf{loop}\ O\ \mathbf{endloop}$$

$$\frac{B_1 \xrightarrow{O} B_1'}{B_1; B_2 \xrightarrow{O} B_1'; B_2} \quad \frac{B_1 \xrightarrow{\delta}, B_2 \xrightarrow{O} B_2'}{B_1; B_2 \xrightarrow{O} B_2'} \quad \frac{B_1 \xrightarrow{\delta}, B_2 \xrightarrow{\delta}}{B_1; B_2 \xrightarrow{\delta} \mathbf{stop}} \quad \frac{B_1 \xrightarrow{O} B_1', B_2 \xrightarrow{O} B_2', O \in \Omega}{B_1 |[\Omega]| B_2 \xrightarrow{O} B_1' |[\Omega]| B_2'} \quad \frac{B_1 \xrightarrow{O} B_1', O \notin \Omega}{B_1 |[\Omega]| B_2 \xrightarrow{O} B_1' |[\Omega]| B_2}$$

$$\frac{B_2 \xrightarrow{O} B_2', O \notin \Omega}{B_1 |[\Omega]| B_2 \xrightarrow{O} B_1 |[\Omega]| B_2'} \quad \frac{B_1 \xrightarrow{\delta} B_1', B_2 \xrightarrow{\delta} B_2'}{B_1 |[\Omega]| B_2 \xrightarrow{\delta} B_1' |[\Omega]| B_2'} \quad \frac{B_1 \xrightarrow{O} B_1'}{B_1 [] B_2 \xrightarrow{O} B_1'} \quad \frac{B_2 \xrightarrow{O} B_2'}{B_1 [] B_2 \xrightarrow{O} B_2'} \quad \frac{B_1 \xrightarrow{O} B_1'}{B_1 [> B_2 \xrightarrow{O} B_1' [> B_2}$$

$$\frac{B_1 \xrightarrow{\delta} B_1'}{B_1 [> B_2 \xrightarrow{\mathbf{i}} \mathbf{null}} \quad \frac{B_2 \xrightarrow{O} B_2'}{B_1 [> B_2 \xrightarrow{O} B_2'} \quad \frac{B_1 \xrightarrow{O} B_1'}{\mathbf{rename}\ R\ \mathbf{in}\ B_1\ \mathbf{endren} \xrightarrow{New(O,R)} \mathbf{rename}\ R\ \mathbf{in}\ B_1'\ \mathbf{endren}}$$

$$\frac{B_1 \xrightarrow{\delta} B_1'}{\mathbf{rename}\ R\ \mathbf{in}\ B_1\ \mathbf{endren} \xrightarrow{\delta} \mathbf{rename}\ R\ \mathbf{in}\ B_1'\ \mathbf{endren}}$$

Figure 1. Original process semantics.

$$\mathbf{null} \xrightarrow{\Delta} \mathbf{null} \quad O \xrightarrow{O} \mathbf{null} \quad \mathbf{loop}\ O\ \mathbf{endloop} \xrightarrow{O} \mathbf{loop}\ O\ \mathbf{endloop}$$

$$\frac{B_1 \xrightarrow{O} B_1'}{B_1; B_2 \xrightarrow{O} B_1'; B_2} \quad \frac{B_1 \xrightarrow{\Delta}, B_2 \xrightarrow{O} B_2'}{B_1; B_2 \xrightarrow{O} B_2'} \quad \frac{B_1 \xrightarrow{\Delta}, B_2 \xrightarrow{\Delta}}{B_1; B_2 \xrightarrow{\Delta} \mathbf{null}} \quad \frac{B_1 \xrightarrow{G} B_1', B_2 \xrightarrow{G} B_2', G \in (\Omega \cup \{\Delta\})}{B_1 |[\Omega]| B_2 \xrightarrow{G} B_1' |[\Omega]| B_2'}$$

$$\frac{B_1 \xrightarrow{O} B_1', O \notin \Omega}{B_1 |[\Omega]| B_2 \xrightarrow{O} B_1' |[\Omega]| B_2} \quad \frac{B_2 \xrightarrow{O} B_2', O \notin \Omega}{B_1 |[\Omega]| B_2 \xrightarrow{O} B_1 |[\Omega]| B_2'} \quad \frac{B_1 \xrightarrow{O} B_1', B_2 \xrightarrow{O} B_2', O \in \Omega}{B_1 [\Omega] B_2 \xrightarrow{O} B_1' [\Omega] B_2'}$$

$$\frac{B_1 \xrightarrow{O} B_1', B_2 \xrightarrow{O}\not\rightarrow, O \in \Omega}{B_1 [\Omega] B_2 \xrightarrow{O} B_1'} \quad \frac{B_2 \xrightarrow{O} B_2', B_1 \xrightarrow{O}\not\rightarrow, O \in \Omega}{B_1 [\Omega] B_2 \xrightarrow{O} B_2'} \quad \frac{B_1 \xrightarrow{O} B_1', O \notin \Omega}{B_1 [\Omega] B_2 \xrightarrow{O} B_1'} \quad \frac{B_2 \xrightarrow{O} B_2', O \notin \Omega}{B_1 [\Omega] B_2 \xrightarrow{O} B_2'}$$

$$\frac{B_1 \xrightarrow{O} B_1', B_2 \xrightarrow{O} B_2', O \in \Omega}{B_1 [\Omega > B_2 \xrightarrow{O} B_1' [\Omega > B_2'} \quad \frac{B_1 \xrightarrow{O} B_1', B_2 \xrightarrow{O}\not\rightarrow, O \in \Omega}{B_1 [\Omega > B_2 \xrightarrow{O} B_1'} \quad \frac{B_2 \xrightarrow{O} B_2', B_1 \xrightarrow{O}\not\rightarrow, O \in \Omega}{B_1 [\Omega > B_2 \xrightarrow{O} B_2'} \quad \frac{B_1 \xrightarrow{O} B_1', O \notin \Omega}{B_1 [\Omega > B_2 \xrightarrow{O} B_1' [\Omega > B_2}$$

$$\frac{B_2 \xrightarrow{O} B_2', O \notin \Omega}{B_1 [\Omega > B_2 \xrightarrow{O} B_2'} \quad \frac{B_1 \xrightarrow{G} B_1'}{\mathbf{rename}\ R\ \mathbf{in}\ B_1\ \mathbf{endren} \xrightarrow{New(G,R)} \mathbf{rename}\ R\ \mathbf{in}\ B_1'\ \mathbf{endren}}$$

Figure 2. Desired process semantics.

can also change the type of a gate, merge a set of gates or split a gate.

We sometimes use "$[]$" or "$|||$" as a prefix operator, where composition of an empty set of processes is supposed to be equivalent to **stop**.

3 The Desired Process Semantics

3.1 Definition

Let $\mathcal{O}$ denote the universe of gates O for actions which we call ordinary actions, as opposed to special-purpose actions introduced in Sect. 4. Ω denotes a subset of $\mathcal{O}$. For a B, $\mathcal{O}(B)$ denotes its gates in $\mathcal{O}$.

We assume that processes in which decisions are to be delayed contain only the following constructs: "**stop**", "**null**", "O", "$B_1; B_2$", "$B_1 |[\Omega]| B_2$", "**loop** O **endloop**", "$B_1 [] B_2$" where neither B_1 nor B_2 can successfully terminate initially, "$B_1 [> B_2$" where B_2 never successfully terminates initially, and "**rename gate** O_1 **is** $O_1' \ldots$ **gate** O_n **is** O_n' **in** B_1 **endren**" where the renaming introduces no gate merging or splitting.

The original semantics of such processes is given in Fig. 1, where δ denotes successful termination, $\mathbf{i}$ denotes an anonymous internal process action and $New(G, R)$ is the new name of a G as generated by renamings R (equals G for the non-renamed G). In the following, let i and i' denote two different members of $\{1, 2\}$.

In Fig. 2, we define the desired modified process semantics. "$B_1 [] B_2$" is enhanced into "$B_1 [\Omega] B_2$", a generalization of "$B_1 [\emptyset] B_2$", i.e. of "$B_1 [] B_2$". As long as B_1 and B_2 are both ready to execute an O in Ω, it can be executed only as their common action and doesn't resolve the choice. Only when a B_i executes an action without co-operation of $B_{i'}$, it becomes the selected alternative. We require that Ω is such that the choice is made before B_1 or B_2 becomes ready to successfully terminate.

Likewise, "$B_1 [> B_2$" is enhanced into "$B_1 [\Omega > B_2$". As long as B_1 and B_2 are both ready to execute an O in Ω, it can be executed only as their common action and doesn't cause disabling of B_1. Only when B_2 executes an action without co-operation of B_1, B_1 is disabled. Analogously, when B_1 executes an O in Ω individually, B_2 is cancelled. We require that Ω is such that the choice is made before B_1 or B_2 becomes ready to successfully terminate. "$B_1 [> B_2$" is now a shorthand for "$B_1 [\emptyset > B_2$", and no longer represents the original "$B_1 [> B_2$", which is equivalent to "**hide** O **in** $(B_1; O) [O > B_2$ **endhide**" with the hidden O present neither in B_1 nor in B_2.

To simplify implementation, we also modify the semantics of successful termination, and consequently of

$$
\begin{array}{l}
D_P[\![\mathbf{stop}(As)]\!] := (|||_{A\in(As\cap P)}\mathbf{loop}\ \delta_A\ \mathbf{endloop})\\
D_P[\![A]\!] := ((|||_{(A'\in P)\wedge I(A,A')}\mathbf{loop}\ \delta_{A'}\ \mathbf{endloop})[> (A; (|||_{A'\in P}\mathbf{loop}\ \delta_{A'}\ \mathbf{endloop})))\\
D_P[\![B_1; B_2]\!]\ \underline{\text{where}}\ (((\mathcal{A}(B_1)\cup P)\cap\mathcal{A}(B_2)) = \emptyset) := (\mathbf{rename}\ \underline{\text{forall}}\ A\in\mathcal{A}(B_2) : \mathbf{gate}\ \delta_A\ \mathbf{is}\ A\ \underline{\text{endfor}}\\
\qquad\qquad\mathbf{in}\ D_{P\cup\mathcal{A}(B_2)}[\![B_1]\!]\ \mathbf{endren}\\
\qquad\qquad|[\mathcal{A}(B_2)\cup\{\delta_A|(A\in P)\}]|D_P[\![B_2]\!])\\
D_P[\![B_1|[As]|B_2]\!] := (D_P[\![B_1]\!]|[As\cup\{\delta_A|(A\in P)\}]|D_P[\![B_2]\!])\\
D_P[\![B_1[]B_2]\!] := (D_P[\![B_1]\!][\{\delta_A|(A\in P)\}]D_P[\![B_2]\!])\\
D_P[\![B_1[>B_2]\!] := (D_P[\![B_1]\!][\{\delta_A|(A\in P)\}>D_P[\![B_2]\!])
\end{array}
$$

Figure 3. Implementation of dependence-based action ordering.

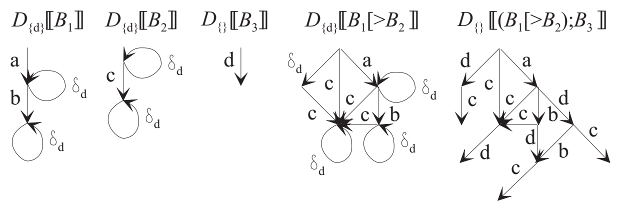

Figure 4. Example implementation of dependence-based action ordering.

$$
\frac{B \xrightarrow{G} B'}{L_N[\![B]\!] \xrightarrow{G(x)} L_N[\![B']\!]}\ \begin{bmatrix} G\in(\mathcal{O}\cup\{\Delta\})\\ x:\mathrm{nat}\end{bmatrix}
$$

$$
\frac{B \xrightarrow{O},\ B \not\xrightarrow{\overline{A}}}{L_N[\![B]\!] \xrightarrow{\overline{O}(x,\mathrm{true})} L_N[\![\mathbf{stop}]\!]}\ \begin{bmatrix} O\in N\\ x:\mathrm{nat}\end{bmatrix}
$$

$$
\frac{B \not\xrightarrow{O},\ B \not\xrightarrow{\overline{A}}}{L_N[\![B]\!] \xrightarrow{\overline{O}(x,\mathrm{false})} L_N[\![\mathbf{stop}]\!]}\ \begin{bmatrix} O\in N\\ x:\mathrm{nat}\end{bmatrix}
$$

Figure 5. Implementation semantics.

sequential and of parallel composition. The idea is to eliminate the need for δ, the only event which has a special status in the original process semantics. So **null** now rather stands for an infinite sequence of special-purpose actions Δ. Unlike δ, Δ can be subject to renaming and optional synchronization, like any other action.

3.2 Applications

3.2.1 Incremental Specification

The generalized choice operator "$[\Omega]$" is a generalization of the process merge operator $\oplus$ defined in [3], which is equivalent to "$[\mathcal{O}]$" and is hence not a generalization of "[]". For the most typical application of late choice, which is incremental system design, $\oplus$ is just the right process merging operator, but as indicated in [3, 6], one has to stay within the universe of deterministic processes. In our setting, the safest way to achieve that is to take care that in every "$B_1|[\Omega]|B_2$", "$B_1[\Omega]B_2$" or "$B_1[\Omega > B_2$", $(\mathcal{O}(B_1)\cap\mathcal{O}(B_2))\subseteq\Omega$.

3.2.2 Dependence-Based Action Ordering

[7] introduced a process algebra supporting dependence-based action ordering. In that algebra, sequential composition of processes is weak, i.e. in any "$B_1; B_2$", mutual independence of an action A_1 in B_1 and an action A_2 in B_2 implies that they commute. Such process composition can also be regarded as parallel composition where every action A in B_2 is synchronized with a special permission δ_A issued by B_2.

Let $\mathcal{A}$ denote the universe of actions A, and $\mathcal{A}(B)$ such actions of a B. All A and δ_A are members of $\mathcal{O}$.

A permission δ_A issued by a "$B_1[]B_2$" (or a "$B_1[> B_2$") is either a common action of B_1 and B_2 or, if a B_i currently doesn't provide it, an action of $B_{i'}$, which thereby becomes the selected alternative. Hence the composition is actually "$B_1[\Omega]B_2$" (or "$B_1[\Omega > B_2$") with Ω consisting of action permissions.

In a "$B_1; B_2$" or a "$B_1|[As]|B_2$" (where $As\subseteq\mathcal{A}$), every δ_A is strictly a common action of B_1 and B_2. The most simple process "$\mathbf{stop}(As)$" provides just permissions δ_A for A in As. An individual action process "A" initially provides just permissions for A' independent from A, while other permissions become available after execution of A.

The $D_P[\![B]\!]$ in Fig. 3 takes an ordinary process B and relaxes the ordering of its actions as much as allowed by the independence relation I. P is the set of actions A for which $D_P[\![B]\!]$ is supposed to generate permissions δ_A. It depends on the context in which B is embedded. If B is not further combined, P is empty. With a restriction on sequential composition (which can be circumvented by gate merging on the highest level), we have been able to keep the generated processes within the scope defined in Fig. 2. Hence they can be translated into E-LOTOS as described in Sect. 4. An example implementation of dependence-based action ordering is given in Fig. 4, where $I(\mathrm{b},\mathrm{d})$ and $I(\mathrm{c},\mathrm{d})$, but not $I(\mathrm{a},\mathrm{d})$.

4 Encoding in E-LOTOS

4.1 Implementation Semantics

For every O, we introduce three new gates, $\overline{O}$, A_O and $\overline{A_O}$. We expect that $L_N[\![B]\!]$, an E-LOTOS encoding of a process B with semantics defined in Fig. 2, behaves as de-

$L_N[\![\mathbf{stop}]\!] := (|||_{O \in N} \mathbf{loop}\ \overline{O}(\mathbf{any}:\mathrm{nat}, !\mathrm{false})\ \mathbf{endloop})$
$L_N[\![\mathbf{null}]\!] := \mathbf{loop}\ \Delta(\mathbf{any}:\mathrm{nat})\ \mathbf{endloop}$
$L_N(O) := ((O(\mathbf{any}:\mathrm{nat}); L_N(\mathbf{null}))[](\![]_{O' \in N}(\overline{O'}(\mathbf{any}:\mathrm{nat}, !(O'=O)); L_N(\mathbf{stop}))))$
$L_N[\![\mathbf{loop}\ O\ \mathbf{endloop}]\!] := (\mathbf{loop}\ O(\mathbf{any}:\mathrm{nat})\ \mathbf{endloop} [> (\![]_{O' \in N}(\overline{O'}(\mathbf{any}:\mathrm{nat}, !(O'=O)); L_N[\![\mathbf{stop}]\!])))$
$L_N[\![\mathbf{rename}\ R\ \mathbf{in}\ B_1\ \mathbf{endren}]\!] :=$
  $\mathbf{rename}\ \underline{\text{forall}}\ \{O|((O \in \mathcal{O}(B_1)) \wedge (New(O,R) \neq O))\} : \mathbf{gate}\ O\ \mathbf{is}\ New(O,R)\ \underline{\text{endfor}}$
  $\underline{\text{forall}}\ \{O|((New(O,R) \in N) \wedge (New(O,R) \neq O))\} : \mathbf{gate}\ \overline{O}\ \mathbf{is}\ \overline{New(O,R)}\ \underline{\text{endfor}}$
  $\mathbf{in}\ L_{\{O|(New(O,R) \in N)\}}[\![B_1]\!]\ \mathbf{endren}$

Figure 6. Implementation of simple processes.

$Split(G(x_1,\ldots,x_m) \rightarrow \{G_0(y_{0,1},\ldots,y_{0,k_0}),\ldots,G_{n-1}(y_{n-1,1},\ldots,y_{n-1,k_{n-1}})\}) :=$
$\underline{\text{forall}}\ z \in \{0,\ldots,n-1\} : \mathbf{gate}\ G(?j:\mathrm{mod\_}z\_n, !x_1,\ldots,!x_m)\ \mathbf{is}\ G_z(!(j \text{ div } n), !y_{z,1},\ldots,!y_{z,k_z})\ \underline{\text{endfor}}$

Figure 7. Gate splitting.

$L_N[\![B_1;B_2]\!] := \mathbf{rename}\ \underline{\text{forall}}\ O \in \mathcal{O}(B_1) : \mathbf{gate}\ A_O\ \mathbf{is}\ O\ \underline{\text{endfor}}\ \underline{\text{forall}}\ O \in N : \mathbf{gate}\ \overline{A_O}\ \mathbf{is}\ \overline{O}\ \underline{\text{endfor}}$
  $\mathbf{in}\ C_1|[\mathcal{O}(B_2) \cup \{\Delta\} \cup \{\overline{O}|(O \in N)\}]|C_2\ \mathbf{endren}$
$C_1 := \mathbf{rename}\ \underline{\text{forall}}\ O \in \mathcal{O}(B_1) : \mathbf{gate}\ O\ \mathbf{is}\ A_O\ \underline{\text{endfor}}\ \underline{\text{forall}}\ O \in N : \mathbf{gate}\ \overline{O}\ \mathbf{is}\ \overline{A_O}\ \underline{\text{endfor}}$
  $Split(\Delta \rightarrow (\mathcal{O}(B_2) \cup \{\Delta\} \cup \{\overline{O}(\mathrm{false})|(O \in N)\} \cup \{\overline{O}(\mathrm{true})|(O \in N)\}))$
  $\mathbf{in}\ L_N[\![B_1]\!]\ \mathbf{endren}$
$C_2 := L_N[\![B_2]\!]$

Figure 8. Implementation of sequential composition.

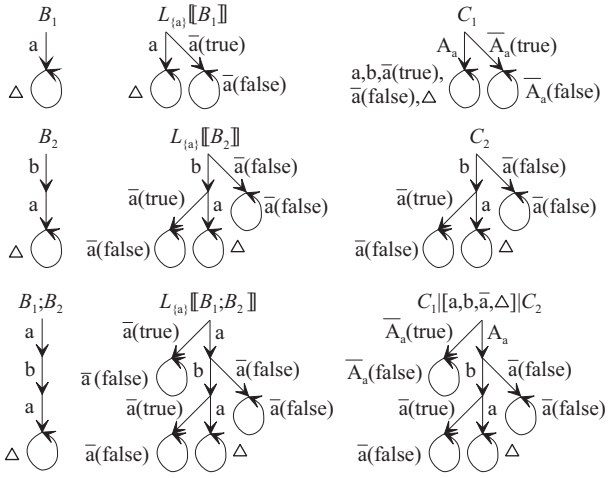

Figure 9. Example implementation of sequential composition.

fined in Fig. 5 (for simple processes, that can be encoded as in Fig. 6):

Every action must carry an auxiliary parameter facilitating its splitting (see Sect. 4.2). The parameter must be of type "nat", i.e. ranging over all natural numbers, including 0.

If an action G, where G is an O or Δ, reduces B to a B', $G(x)$ (where x is the gate-splitting parameter) must reduce $L_N[\![B]\!]$ to $L_N[\![B']\!]$ (or any other process with the same behaviour).

For any O in N, an auxiliary action $\overline{O}(x,y)$ (where x is the gate-splitting parameter) must reduce $L_N[\![B]\!]$ to a process equivalent to $L_N[\![\mathbf{stop}]\!]$, thereby modelling disruption of B upon an O executed by a competing process. Hence N is supposed to be the set of all O potentially disruptive for B. It depends on the context in which B is embedded. If B is not further combined, N is empty. The second parameter y must be a boolean value indicating whether $L_N[\![B]\!]$ could as well execute an $O(x)$ in its present state, i.e. indicating the current (un)readiness of B for O. When $L_N[\![B]\!]$ starts indicating successful termination of B by Δ, it stops executing $\overline{O}$ actions.

Auxiliary actions can be removed from an $L_N[\![B]\!]$ by a concurrent process constraining its behaviour to ordinary actions and Δ [9]. Auxiliary action parameters can be removed by action renaming.

4.2 Specification Style

From various well-known specification styles (see [9]), we choose the constraint-oriented style. In this style, a process is defined by a set of constraints on its actions, represented as concurrent processes synchronized on the actions they collectively control.

In every $L_N[\![B_1 * B_2]\!]$, where "$*$" is some composition operator, there is a constraint C_i for every B_i (see Figs. 8, 10, 12 and 13). Every C_i is an $L_{N_i}[\![B_i]\!]$, where N_i is N enhanced to meet the needs of $B_{i'}$ with respect to "$*$". Every action of $L_{N_i}[\![B_i]\!]$ is split into all kinds of actions of $L_N[\![B_1 * B_2]\!]$ in which it might have to participate.

In E-LOTOS, gate splitting is renaming of actions on a gate according to the data they carry. Therefore we furnish every gate with an auxiliary parameter serving for its splitting. In Fig. 7, where type "mod_z_n" denotes natural numbers j with $((j \bmod n) = z)$,

$L_N[\![B_1 |[\Omega]| B_2]\!] :=$ **rename** forall $O \in N :$ **gate** $\overline{O}(?x : \text{nat}, ?y : \text{bool}, \textbf{etc})$ **is** $\overline{O}(!x, !y)$ endfor
in $C_1 |[\Omega \cup \{\Delta\} \cup \{\overline{O} | (O \in N)\}]| C_2$ **endren**
$C_i :=$ **rename** forall $O \in N : Split_i(O)$ endfor
$Split(\Delta \rightarrow (\{\overline{O}(\text{false}, -i) | (O \in N)\} \cup \{\overline{O}(\text{true}, -i) | (O \in (N \backslash \Omega))\} \cup \{\Delta\}))$
in $L_N[\![B_i]\!]$ **endren**
$Split_i(O) :=$ if $O \in \Omega$ then $Split(\overline{O}(\text{false}) \rightarrow \{\overline{O}(\text{false}), \overline{O}(\text{false}, i), \overline{O}(\text{false}, -i')\})$
$Split(\overline{O}(\text{true}) \rightarrow \{\overline{O}(\text{true}), \overline{O}(\text{false}, i'), \overline{O}(\text{false}, -i')\})$
else $Split(\overline{O}(\text{false}) \rightarrow \{\overline{O}(\text{false}), \overline{O}(\text{true}, i'), \overline{O}(\text{false}, -i')\})$
$Split(\overline{O}(\text{true}) \rightarrow \{\overline{O}(\text{true}), \overline{O}(\text{true}, i), \overline{O}(\text{true}, -i')\})$ endif

Figure 10. Implementation of parallel composition.

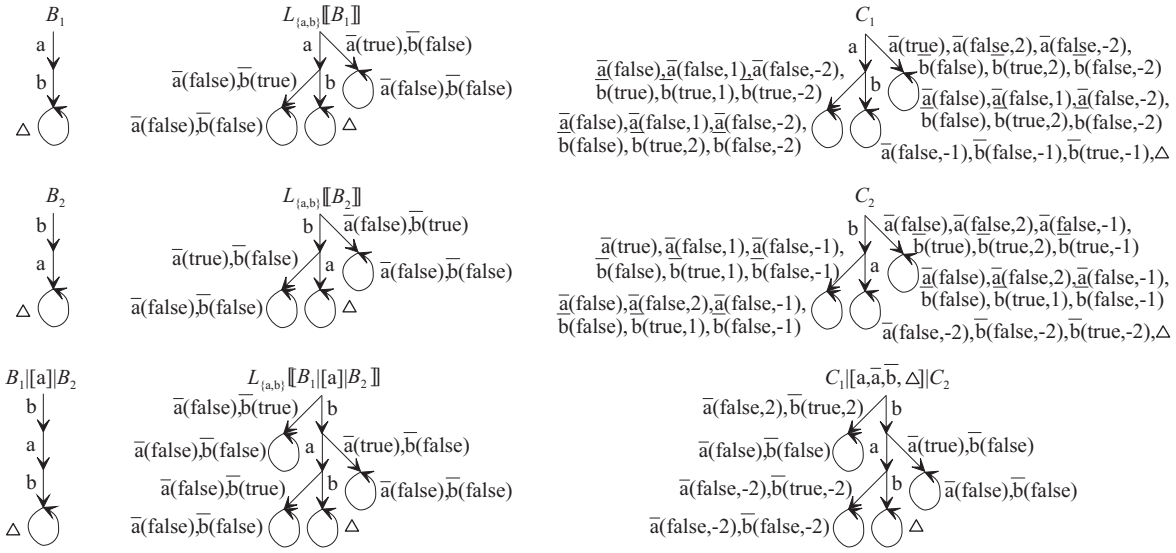

Figure 11. Example implementation of parallel composition.

$L_N[\![B_1 [\Omega] B_2]\!] :=$ **rename** forall $O \in (\mathcal{O}(B_1) \cup \mathcal{O}(B_2)) :$ **gate** $O(?x : \text{nat}, \textbf{etc})$ **is** $O(!x)$ endfor
forall $O \in N :$ **gate** $\overline{O}(?x : \text{nat}, ?y : \text{bool}, \textbf{etc})$ **is** $\overline{O}(!x, !y)$ endfor
in $C_1 |[\mathcal{O}(B_1) \cup \mathcal{O}(B_2) \cup \{\overline{O} | (O \in N)\}]| C_2$ **endren**
$C_i :=$ **rename** forall $O \in \mathcal{O}(B_i) :$ if $O \in \Omega$ then $Split(O \rightarrow \{O, O(i)\})$ else $Split(O \rightarrow \{O(i)\})$ endif endfor
forall $O \in (N \backslash \mathcal{O}(B_{i'})) : Split_i(O)$ endfor forall $O \in (\mathcal{O}(B_{i'}) \backslash N) : Split'_i(O)$ endfor
forall $O \in (\mathcal{O}(B_{i'}) \cap N) : Split''_i(O)$ endfor
in $L_{N \cup \mathcal{O}(B_{i'})}[\![B_i]\!]$ **endren**
$Split_i(O) := Split(\overline{O}(\text{false}) \rightarrow \{\overline{O}(\text{false}), \overline{O}(\text{true}, i')\})\ Split(\overline{O}(\text{true}) \rightarrow \{\overline{O}(\text{true}), \overline{O}(\text{true}, i)\})$
$Split'_i(O) := Split(\overline{O}(\text{false}) \rightarrow \{O(i')\})$
if $O \in \Omega$ then $Split(\overline{O}(\text{true}) \rightarrow \{\overline{O}(\text{true}, i)\})$ else $Split(\overline{O}(\text{true}) \rightarrow \{O(i')\})$ endif
$Split''_i(O) := Split(\overline{O}(\text{false}) \rightarrow \{O(i'), \overline{O}(\text{false}), \overline{O}(\text{true}, i')\})$
if $O \in \Omega$ then $Split(\overline{O}(\text{true}) \rightarrow \{\overline{O}(\text{true}), \overline{O}(\text{true}, i)\})$
else $Split(\overline{O}(\text{true}) \rightarrow \{O(i'), \overline{O}(\text{true}), \overline{O}(\text{true}, i)\})$ endif

Figure 12. Implementation of choice.

we define a function $Split$ generating the renamings necessary for splitting a gate originally defined as a $G(x_1, \ldots, x_m)$ into gates originally defined as $G_0(y_{0,1}, \ldots, y_{0,k_0}), \ldots, G_{n-1}(y_{n-1,1}, \ldots, y_{n-1,k_{n-1}})$ (where a $G()$ may be shortened into G). In our graphical examples (Figs. 9, 11, 14 and 15), the gate-splitting parameter of actions is omitted. Where a transition is labelled with more than one action, they should be interpreted as alternatives.

5 Conclusion

We have demonstrated that for processes executing only ordinary observable actions, late choice and dependence-based action ordering can be easily encoded in the original E-LOTOS. A more detailed description of the method is given in [4]. We have employed similar ideas as in [5], where we describe a very general method for E-LOTOS-based specification of further non-standard forms of pro-

$L_N[\![B_1[\Omega > B_2]\!] :=$ **rename** forall $O \in (\mathcal{O}(B_1)\backslash\Omega)$: **gate** A_O **is** O endfor
forall $O \in ((\mathcal{O}(B_1) \cap \Omega) \cup \mathcal{O}(B_2)))$: **gate** $O(?x : \mathrm{nat}, \mathbf{etc})$ **is** $O(!x)$ endfor
forall $O \in N$: **gate** $\overline{O}(?x : \mathrm{nat}, ?y : \mathrm{bool}, \mathbf{etc})$ **is** $\overline{O}(!x, !y)$ endfor
in $C_1 |[(\mathcal{O}(B_1) \cap \Omega) \cup \mathcal{O}(B_2) \cup \{\overline{O} | (O \in N\}]| C_2$ **endren**

$C_1 :=$ **rename** forall $O \in \mathcal{O}(B_1)$: if $O \in \Omega$ then $Split(O \to \{O, O(1)\})$ else **gate** O **is** A_O endif endfor
forall $O \in (N\backslash\mathcal{O}(B_2))$: $Split_1(O)$ endfor forall $O \in (\mathcal{O}(B_2)\backslash N)$: $Split_1'(O)$ endfor
forall $O \in (\mathcal{O}(B_2) \cap N)$: $Split_1''(O)$ endfor
in $L_{N \cup \mathcal{O}(B_2)}[\![B_1]\!]$ **endren**

$C_2 :=$ **rename** forall $O \in \mathcal{O}(B_2)$: if $O \in \Omega$ then $Split(O \to \{O, O(2)\})$ else $Split(O \to \{O(2)\})$ endif endfor
forall $O \in (N\backslash(\mathcal{O}(B_1) \cap \Omega))$: $Split_2(O)$ endfor forall $O \in ((\mathcal{O}(B_1) \cap \Omega)\backslash N)$: $Split_2'(O)$ endfor
forall $O \in (\mathcal{O}(B_1) \cap \Omega \cap N)$: $Split_2''(O)$ endfor
in $L_{N \cup (\mathcal{O}(B_1) \cap \Omega)}[\![B_2]\!]$ **endren**

$Split_i(O)$, $Split_i'(O)$ and $Split_i''(O)$ as in Fig. 12.

Figure 13. Implementation of disabling.

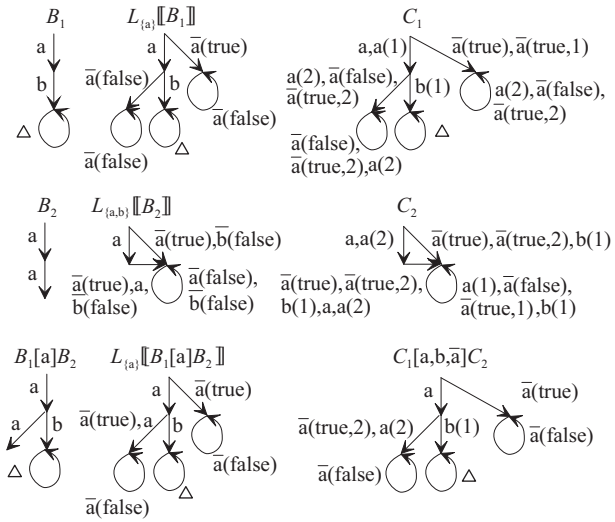

Figure 14. Example implementation of choice.

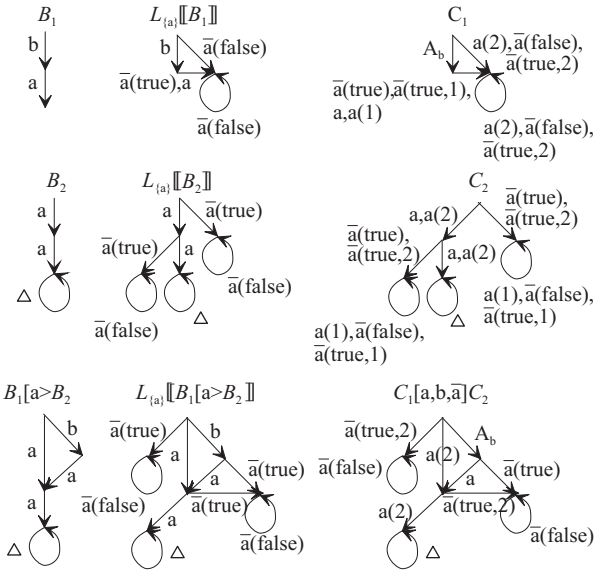

Figure 15. Example implementation of disabling.

cess composition.

6 References

[1] T. Bolognesi and E. Brinksma, "Introduction to the ISO specification language LOTOS," *Computer Networks and ISDN Systems*, vol. 14, no. 1, pp. 25–59, 1987.

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

[3] H. Ichikawa, K. Yamanaka, and J. Kato, "Incremental specification in LOTOS," in *Protocol Specification, Testing, and Verification*, X, eds. L. Logrippo, R. L. Probert, and H. Ural, pp. 183–196, North-Holland, 1990.

[4] M.Kapus-Kolar, "Specifying Late Decisions in a Sublanguage of E-LOTOS," Jožef Štefan Institute technical report #8931, 2004.

[5] M. Kapus-Kolar, "A generalization of the E-LOTOS renaming operator: a convenience for specification of new forms of process composition," *Computer Standards & Interfaces*, vol. 26, no. 6, pp. 549–563, 2004.

[6] F. Khendek and G. von Bochmann, "Merging behavior specifications," *Formal Methods in System Design*, vol. 6, no. 3, pp. 259–295, 1995.

[7] A. Rensink and H. Wehrheim, "Process algebra with action dependencies," *Acta Informatica*, vol. 38, no. 3, pp. 155–234, 2001.

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

[9] C. A. Vissers, G. Scollo, M. van Sinderen, and H. Brinksma, "Specification styles in distributed systems design and verification," *Theoretical Computer Science*, vol. 89, no. 1, pp. 179–206, 1991.

Monika Kapus-Kolar received the B.S. degree in electrical engineering from the University of Maribor, Slovenia, in 1981, and the M.S. and Ph.D. degrees in computer science from the University of Ljubljana, Slovenia, in 1984 and 1989, respectively. Since 1981 she has been with the Jožef Stefan Institute in Ljubljana. Her current research interests include formal specification techniques and methods for distributed systems development.