A generalization of the E-LOTOS renaming operator: a convenience for specification of new forms of process composition

M. Kapus-Kolar

Jožef Stefan Institute, POB 3000, SI-1001 Ljubljana, Slovenia

Received 2 December 2003; revised 25 February 2004

Abstract

E-LOTOS is one of the standard languages for formal specification of real-time concurrent and reactive systems. As it is process-algebraic, its expressive power lies in its process-composition operators. Of course, not all forms of composition can be directly expressed in the language. In the most difficult cases, one typically resorts to the constraint-oriented specification style. We demonstrate that a slight enhancement of the E-LOTOS renaming operator would make the specification style even more powerful. As an example, we show how to specify choice, parallel composition and disabling enhanced with action priorities.

Key words: Formal methods, E-LOTOS, Constraint-oriented specification, Event renaming, Action priorities

1. Introduction

E-LOTOS [6,10], an enhanced successor of LOTOS [5,1], is one of the standard languages for formal specification of real-time concurrent and reactive systems. An E-LOTOS specification characterizes a process by its readiness to engage into various kinds of atomic instantaneous events, where an event might be

- a $G(RN)$, i.e. an interaction of the process with its environment, where G is the gate on which it occurs and RN is the data record it carries,
- an $X(RN)$, i.e. a signal issued by the process,
- an **i**, i.e. an internal process action, or
- a $\delta(RN)$, i.e. successful termination of the process.

The usual dynamic semantics of an E-LOTOS process is that of interleaving events. A process is characterized by its labelled transition system (LTS), where an individual transition represents either an individual event or idling, i.e. passing of time. For convenience, we will assume that time is discrete, so that every time step is a "**tick**". Events of type **i**, $\delta(RN)$ and $X(RN)$ are by definition urgent, i.e. cannot have idling as an alternative.

To incorporate a process into a particular environment, it is sometimes necessary to relabel its LTS, i.e. to change a $G(RN)$ into a $G'(RN')$, or an $X(RN)$ into an $X'(RN')$. That is facilitated by the E-LOTOS renaming operator. Unfortunately, the current semantics of the operator is unnecessarily rigid. In Sections 2 and 3, we propose a new se-

Email address: `monika.kapus-kolar@ijs.si` (M. Kapus-Kolar).

mantics which makes the operator much more easy to understand and convenient to use. As demonstrated in Section 4, the enhanced operator is particularly helpful in constraint-oriented specification of non-standard forms of process composition.

2. From gate renaming to action renaming

Consider a process B' defined as
"**rename gate** G(!true : bool) **is** G'!1
in B **endren**",
where process B is defined as "G **any** : bool", i.e. represents an interaction on gate G carrying any boolean value. As B executes either G(true) or G(false), one would expect that B' executes either $G'(1)$ or G(false). Wrong! According to the current E-LOTOS semantics, the renaming makes G(false) non-executable, because gate G is being renamed (i.e. actions on the gate are being renamed), but there is no name defined for G(false). In other words, a proper specification of the intended behaviour would be
"**rename gate** G(!true : bool) **is** G'!1
gate G(!false : bool) **is** G!false
in B **endren**".
There are cases where such complication is most inconvenient. For example, when correcting a
"**rename gate** G(!0 : nat) **is** G'!false
in G **any** : nat **endren**",
where "nat" denotes natural numbers, one has to specify non-renaming for every $G(x)$ with $(x > 0)$.

In any case, we observe that one naturally tends see the E-LOTOS renaming operator as an action-renaming operator, and not as a gate-renaming operator. That is most evident in cases where gates are split (as in the above examples) or merged (as, for example, in
"**rename gate** G_1 **is** G **gate** G_2 **is** G
in B **endren**").
Gate splitting and gate merging are operations which are beyond gate renaming, but (assuming that there is a name specified for every action on the gates which are being renamed) not beyond action renaming.

While actions are a primary concept in E-LOTOS, a gate is often given no interpretation beyond being the first item in the tuple of values representing a particular action. For not every process specified represents an architectural component. Often it is just a part of such a component, or an abstraction of its behaviour, or even just another constraint governing its behaviour.

For the above reasons, we propose that E-LOTOS renaming is rather understood as action renaming, meaning that when there is no new name specified for a particular action, the action remains executable in its original form.

The same policy should be taken for renaming of signals: If there is no new name for a signal, the signal remains as it is. For blocking of a signal is even worse than blocking of an action, because signals are in E-LOTOS by definition urgent.

For the present, however, it would be convenient to have the new semantics just as an enhancement to the existing E-LOTOS renaming operator. That can be achieved by a slight syntactic enhancement. The operator currently combines statements of the forms "**gate** $G \ldots$ **is** $\ldots$" and "**signal** $X \ldots$ **is** $\ldots$". If such a statement is used for a G or an X, that means that one wants for G or X the original renaming semantics. If the new renaming semantics is desired for a G or an X, one should use different keywords, for example statements of the forms "**action** $G \ldots$ **is** $\ldots$" and "**sgn** $X \ldots$ **is** $\ldots$", as we do in the rest of the paper.

3. From gate splitting to action splitting

The generic construct for specifying a visible action in E-LOTOS is "$GP_1@P_2[E]$". An action satisfies the specification provided that
- it is on gate G,
- the data it carries matches pattern P_1,
- its execution time measured relatively to the moment when it is logically enabled matches pattern P_2, and
- its data and its execution time satisfy predicate E.

In a general case, a "$GP_1@P_2[E]$" (e.g. a "G **any** : bool") is satisfiable by many different $G(RN)$ (in the example, by G(true) or G(false)). Hence action splitting is not unknown in E-LOTOS.

On the other hand, it seems that there has been an agreement that the E-LOTOS renaming operator must not introduce action splitting (see for example [2]), although we have not been able to detect such a statement in [6]. It must certainly not introduce signal splitting, and that is not stated in [6], either. For observe the process

"**trap exception** X_1 **is** G_1@!1 **endexn**
exception X_2 **is** G_2@!1 **endexn**
in rename signal X **is** X_1 **signal** X **is** X_2
in raise X **endren endtrap**",

structured as follows:

- The innermost process raises exception X, i.e. issues signal X and blocks the system.
- The renaming operator splits signal X so that it can be issued either as an X_1 or as an X_2. Hence the LTS of the resulting process has branches X_1 and X_2.
- The exception-trapping operator replaces the two exception signals with their corresponding handlers "G_1@!1" and "G_2@!1". The LTS of the resulting process has alternative branches "**tick**, G_1, δ" and "**tick**, G_2, δ", the behaviours of the two handlers. Hence in the initial state of the process, there are two **tick** transitions, implying that the choice is resolved by the flow of time.

The process is illegal in E-LOTOS, for time nondeterminism is in E-LOTOS a taboo. In the example, time non-determinism emerges because of the trapping operator, but its seed is in the signal splitting.

For actions, trapping is not defined, hence their splitting can do no harm. Therefore we define that the new semantics for action renaming allows action splitting, because in E-LOTOS with action splitting forbidden, language users have unnecessary troubles:

- Specifying action renaming, one must be careful not to define more than one new name per action.
- For every action, one must plan its "splittings" in advance, to furnish it with auxiliary parameters facilitating the splittings. For example, if an action G is to experience a binary split, "G **any** : bool" would be its adequate specification.

The generic syntax for gate renaming is "**gate** $G(IPL)$ **is** $G'P$", where P, i.e. the data carried by the new gate, is a function of IPL, i.e. of the data carried by the old gate. For action renaming, we propose a slightly enhanced syntax, namely "**action** $G(IPL)$ **is** $G'P[E]$". The expression is a shorthand for the set of all such renamings of a $G(RN)$ into a $G'(RN')$ that

- RN matches IPL,
- RN' matches P, and
- the pair (RN, RN') satisfies E.

The predicate E, whose default value is true, has a similar role as in the specification of an individual action. It can help in two ways:

- It facilitates specification of which actions on gate G to rename. For example, an
  "**action** $G(?V : \text{int})$ **is** $G'!\text{true}[V > 0]$"
  concisely specifies renaming of all actions on gate G carrying a positive integer.
- It facilitates specification of the new names. For example, an
  "**action** $G(?V : \text{nat})$
  is $G?V' : \text{nat}[(V' \bmod V) = 0]$"
  concisely specifies that a $G(N)$ may be executed as any $G(N * M)$.

The generic syntax for signal renaming is "**signal** $X(IPL)$ **is** $X'E$", where E, i.e. the data carried by the new signal, is a function of IPL, i.e. of the data carried by the old signal. For the new kind of signal renaming, we propose a slightly enhanced syntax, namely "**sgn** $X(IPL)[E']$ **is** $X'E$". The expression is a shorthand for the set of all such renamings of an $X(RN)$ into an $X'(RN')$ that

- RN matches IPL and satisfies predicate E', and
- RN' is the value of E computed on RN.

The predicate E', whose default value is true, facilitates specification of which X signals to rename. For example, a

"**sgn** $X(?V : \text{int})[V > 0]$ **is** $X'(V + 1)$"

concisely specifies renaming of all signals X carrying a positive integer.

The new renaming semantics defines that a $G'(RN')$ is a new name for a $G(RN)$ if there is at least one "**action**..." statement specifying that. Likewise, an $X'(RN')$ is a new name for an $X(RN)$ if there is at least one "**sgn**..." statement specifying that. The only restriction for combining renaming statements is that signal splitting is forbidden.

Let us compare renaming restricted to "**gate**..." and "**signal**..." statements, i.e. the old renaming operator, with renaming restricted to "**action**..." and "**sgn**..." statements, i.e. the new renaming operator. The new operator has several advantages:

- Most importantly, it specifies nothing beyond event renaming, while the old operator, if not used with sufficient care, might make some events unexpectedly non-executable.
- It gives a specifier complete freedom on what to rename and how to rename, as long as signal splitting is not introduced.
- With the additional predicates, it facilitates concise specification of large sets of renamings.
- It facilitates action splitting, which is, as demonstrated in Section 4, an extremely useful concept. With that feature, E-LOTOS would gain an action-splitting operator even more general than the "multiple labelling" operator of CSP [4].

We observe that the new operator is much more easy to understand and convenient to use than the old one. The only sensible motivation for using the old operator would be for blocking of some events. But visible actions of a process can be blocked also by other means, by synchronizing the process with processes acting as additional constraints. As for signals, they should better never be blocked, as they have been conceived as unblockable events.

4. Action splitting in specification of new forms of process composition

4.1. *Object-oriented constraint-oriented specification*

Action splitting is a step towards polymorphism, which is an important concept in object-oriented specification. An E-LOTOS process is an object characterized by its ability to interact with its environment. By renaming of actions and/or signals, a process can be adapted for proper operation in an environment for which it was not originally intended. If it is a dynamically changing environment, action renaming must provide adequate action names for each of the possible situations, i.e. action splitting might be necessary.

As an example, take a system of processes communicating through synchronous broadcast. In such a system, each communication event synchronizes a message transmitter and all the processes currently ready to receive the message. Hence when a process executes a transmission, it must be ready to execute it either on its own or in co-operation with any group of the remaining processes. Since in E-LOTOS, the degree of synchronization of an action with the process environment directly depends on its name, multiple degrees of synchronization per action can be specified only as action splitting. With this concept, we have indeed been able to model such systems in E-LOTOS [7].

Parallel composition of processes communicating through synchronous broadcast can be expressed in E-LOTOS only indirectly. Although the expressive power of a process-algebraic specification language lies in its process-composition operators, one can hardly expect a language to have operators for all forms of composition. For even if it is from time to time systematically enhanced with new operators for newly identified practically interesting forms of composition, the enhancements must not be too extensive, for otherwise the language would become too complex.

Hence a good specification language provides not only a handy set of process-composition operators, but also means for easy specification of those forms of composition which cannot be expressed directly. Systematic specification of a new form of process composition typically bases on some specific specification style. E-LOTOS, like LOTOS, supports many different specification styles [11]. Action splitting is particularly important in the constraint-oriented style, for this style extensively uses the parallel composition operator, i.e. the operator describing interconnection of processes.

In the constraint-oriented style, one specifies the legal behaviours of a system by a set of constraints, in LOTOS or E-LOTOS modelled as concurrent processes synchronized on the actions they collectively control. Although a set of constraints is primarily a set of logical predicates in conjunction, the constraint-oriented style can be used in an object-oriented way, by giving each object and each subobject its individual constraint. If the approach is combined with action splitting, it becomes a powerful method for specification of non-standard forms of process composition.

To illustrate the power of the object-oriented style, and thereby the usefulness of action splitting, we in the following show how one can use E-LOTOS with enhanced renaming for encoding a small Basic-LOTOS-like language in which the operators of choice, parallel composition and disabling are enhanced with action priorities. The description is based on ideas from [8] and [9]. Along with the example, we provide a detailed discussion on how to indirectly specify new forms of process composition, therefore the reader is invited to give the example a careful consideration.

4.2. *An instructive example*

4.2.1. *Problem statement*

We begin with a small Basic-LOTOS-like sublanguage of E-LOTOS. Below we present its constructs and their untimed dynamic semantics:

– "**block**" denotes a time block, i.e. a process with no steps at all.
– "**stop**" denotes an idling process.
– "**null**" denotes successful termination:

$$\mathbf{null} \xrightarrow{\delta} \mathbf{block}$$

– "G" denotes an individual untimed action on a gate G:

$$G \xrightarrow{G} \mathbf{null}$$

In the following, let μ denote $\mathbf{i}$ or a G, while μ^+ denotes δ or a μ.

– "$B_1; B_2$" denotes sequential composition of processes B_1 and B_2:

$$\frac{B_1 \xrightarrow{\mu} B_1'}{B_1; B_2 \xrightarrow{\mu} B_1'; B_2} \qquad \frac{B_1 \xrightarrow{\delta} B_1', B_2 \xrightarrow{\mu} B_2'}{B_1; B_2 \xrightarrow{\mu} \mathbf{null}; B_2'}$$

$$\frac{B_1 \xrightarrow{\delta} B_1', B_2 \xrightarrow{\delta} B_2'}{B_1; B_2 \xrightarrow{\delta} \mathbf{block}}$$

– "$B_1[]B_2$", where neither B_1 nor B_2 has δ as an initial event, denotes a process behaving as B_1 or as B_2, where the choice is made upon the first event:

$$\frac{B_1 \xrightarrow{\mu} B_1'}{B_1[]B_2 \xrightarrow{\mu} B_1'} \qquad \frac{B_2 \xrightarrow{\mu} B_2'}{B_1[]B_2 \xrightarrow{\mu} B_2'}$$

– "$B_1|[\Gamma]|B_2$", where Γ is a set of gates, denotes processes B_1 and B_2 running in parallel and synchronized on gates in Γ and on δ:

$$\frac{B_1 \xrightarrow{\mu} B_1'}{B_1|[\Gamma]|B_2 \xrightarrow{\mu} B_1'|[\Gamma]|B_2}\ [\mu \notin \Gamma]$$

$$\frac{B_2 \xrightarrow{\mu} B_2'}{B_1|[\Gamma]|B_2 \xrightarrow{\mu} B_1|[\Gamma]|B_2'}\ [\mu \notin \Gamma]$$

$$\frac{B_1 \xrightarrow{\mu^+} B_1', B_2 \xrightarrow{\mu^+} B_2'}{B_1|[\Gamma]|B_2 \xrightarrow{\mu^+} B_1'|[\Gamma]|B_2'}\ [\mu^+ \in (\Gamma \cup \{\delta\})]$$

– "$B_1[>B_2$", where B_2 does not have δ as an initial event, denotes a process which basically executes B_1, but as long as B_1 does not successfully terminate, might start executing B_2 instead (disabling):

$$\frac{B_1 \xrightarrow{\mu} B_1'}{B_1[>B_2 \xrightarrow{\mu} B_1'[>B_2} \qquad \frac{B_1 \xrightarrow{\delta} B_1'}{B_1[>B_2 \xrightarrow{\mathbf{i}} \mathbf{null}}$$

$$\frac{B_2 \xrightarrow{\mu} B_2'}{B_1[>B_2 \xrightarrow{\mu} B_2'}$$

We would like to enhance the operators "[]", "|[Γ]|" and "[>" with action priorities and then conceive a compositional transformation which would take a process specified in the enhanced language and encode its behaviour in E-LOTOS. The transformation will implement every process B by a constraint $C[\![B]\!]$, hence the language being implemented must satisfy the following restrictions:

– There must be no B leading to a time block, for $C[\![B]\!]$ is supposed to represent a predicate, i.e. a *timeless* entity.
– No B may ever be deleted, for the behaviour of a system is supposed to be represented by a *static* set of constraints $C[\![B]\!]$.
– Every event must be *synchronizable*, so that conjunction of constraints can be expressed as synchronization of processes $C[\![B]\!]$.
– Every event must be *renamable*, so that it can be adapted to the current context.
– If a B is a composition of a B_1 and a B_2, it must have no events besides those of B_1 and B_2, for otherwise composition of $C[\![B_1]\!]$ and $C[\![B_2]\!]$ would have to be more complicated than ordinary parallel composition.

Obviously, the adopted language must be amended in several ways:

- “**block**” must not be a part of the language.
- δ is an event which leads to a time block and is not renamable. Therefore we replace it with an infinite series of actions with a reserved name Δ. Hence we will have ordinary gates O and a special gate Δ.
- The decision of a “$B_1[>B_2$” to successfully terminate because B_1 has become ready for a δ is represented by an internal action. The action is unsynchronizable and not an action of B_1 or B_2. In E-LOTOS, it prevents time non-determinism which might occur when the δ is trapped. If we replace δ by Δ, the **i** no longer introduces a trappable event, so it may be omitted.
- In the definitions of choice and disabling, there are cases where B_1 deletes B_2, or vice versa. Instead of being deleted, processes should rather be forced into idling.

A language implementing all the above suggestions would comprise the following constructs, where L denotes Δ or an O, and Ω denotes a set of O:

- idling “**stop**”
- successful termination:

$$\mathbf{null} \xrightarrow{\Delta} \mathbf{null}$$

- an individual untimed action on a gate O:

$$O \xrightarrow{O} \mathbf{null}$$

- sequential composition:

$$\frac{B_1 \xrightarrow{O} B_1'}{B_1;B_2 \xrightarrow{O} B_1';B_2} \qquad \frac{B_1 \xrightarrow{\Delta} B_1', B_2 \xrightarrow{L} B_2'}{B_1;B_2 \xrightarrow{L} B_1';B_2'}$$

- choice:

$$\frac{B_1 \xrightarrow{L} B_1'}{B_1[]B_2 \xrightarrow{L} B_1'[]\mathbf{stop}} \qquad \frac{B_2 \xrightarrow{L} B_2'}{B_1[]B_2 \xrightarrow{L} \mathbf{stop}[]B_2'}$$

- parallel composition:

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

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

$$\frac{B_1 \xrightarrow{L} B_1', B_2 \xrightarrow{L} B_2'}{B_1|[\Omega]|B_2 \xrightarrow{L} B_1'|[\Omega]|B_2'}\,[L \in (\Omega \cup \{\Delta\})]$$

- disabling:

$$\frac{B_1 \xrightarrow{O} B_1'}{B_1[>B_2 \xrightarrow{O} B_1'[>B_2} \qquad \frac{B_1 \xrightarrow{\Delta} B_1'}{B_1[>B_2 \xrightarrow{\Delta} B_1'[>\mathbf{stop}}$$

$$\frac{B_2 \xrightarrow{L} B_2'}{B_1[>B_2 \xrightarrow{L} \mathbf{stop}[>B_2'}$$

In the above language, Δ may be in a decisive position, just as δ may be in LOTOS, while E-LOTOS is in this respect not compatible with LOTOS. On the other hand, the language inherits from E-LOTOS its nice sequential composition operator, which is more natural than the LOTOS operator of enabling.

Now we are ready to introduce *action priorities.* We enhance every operator “[]”, “|[Ω]|” or “[>” with its own priority function Π, i.e. into a “[Π]”, “|[$\Omega|\Pi$]|” or “[Π>”, respectively.

Such an operator combines a B_1 and a B_2 into a B. Let Λ_1 denote the initial actions L of B_1, Λ_2 such actions of B_2, and Λ the gates on which B_1 and B_2 are synchronized. Λ equals $(\Omega \cup \{\Delta\})$ for an “|[Ω]|”, and is empty for “[]” and “[>”.

Function Π takes Λ_1, Λ_2 and Λ, and computes a triplet $(First(B), Second(B), Both(B))$ meaning that an L is a legal initial step of B provided that

- it is in $First(B)$ and executed individually by B_1, or
- it is in $Second(B)$ and executed individually by B_2, or
- it is in $Both(B)$ and executed by B_1 and B_2 in co-operation.

To facilitate implementation of a wide range of different priority schemes, we introduce for Π only the following requirements:

- $First(B) \subseteq (\Lambda_1 \backslash \Lambda)$
- $Second(B) \subseteq (\Lambda_2 \backslash \Lambda)$
- $Both(B) \subseteq (\Lambda_1 \cap \Lambda_2 \cap \Lambda)$
- $(First(B) \cup Second(B) \cup Both(B))$ may be empty only if $((\Lambda_1 \backslash \Lambda) \cup (\Lambda_2 \backslash \Lambda) \cup (\Lambda_1 \cap \Lambda_2 \cap \Lambda))$ is empty, because from a non-empty set of the potentially possible initial steps, Π must select at least one.

The enhanced semantics of choice, parallel composition and disabling is the following:

- choice ($B = B_1[\Pi]B_2$):

$$\frac{B_1 \xrightarrow{L} B_1'}{B_1[\Pi]B_2 \xrightarrow{L} B_1'[\Pi]\mathbf{stop}} \; [L \in First(B)]$$

$$\frac{B_2 \xrightarrow{L} B_2'}{B_1[\Pi]B_2 \xrightarrow{L} \mathbf{stop}[\Pi]B_2'} \; [L \in Second(B)]$$

– parallel composition ($B = B_1|[\Omega|\Pi]|B_2$):

$$\frac{B_1 \xrightarrow{O} B_1'}{B_1|[\Omega|\Pi]|B_2 \xrightarrow{O} B_1'|[\Omega|\Pi]|B_2} \; [O \in First(B)]$$

$$\frac{B_2 \xrightarrow{O} B_2'}{B_1|[\Omega|\Pi]|B_2 \xrightarrow{O} B_1|[\Omega|\Pi]|B_2'} \; [O \in Second(B)]$$

$$\frac{B_1 \xrightarrow{L} B_1', B_2 \xrightarrow{L} B_2'}{B_1|[\Omega|\Pi]|B_2 \xrightarrow{L} B_1'|[\Omega|\Pi]|B_2'} \; [L \in Both(B)]$$

– disabling ($B = B_1[\Pi > B_2)$:

$$\frac{B_1 \xrightarrow{O} B_1'}{B_1[\Pi > B_2 \xrightarrow{O} B_1'[\Pi > B_2} \; [O \in First(B)]$$

$$\frac{B_1 \xrightarrow{\Delta} B_1'}{B_1[\Pi > B_2 \xrightarrow{\Delta} B_1'[\Pi > \mathbf{stop}} \; [\Delta \in First(B)]$$

$$\frac{B_2 \xrightarrow{L} B_2'}{B_1[\Pi > B_2 \xrightarrow{L} \mathbf{stop}[\Pi > B_2'} \; [L \in Second(B)]$$

4.2.2. *Implementation semantics*

Pure object-oriented specification style requires that a $C[\![B]\!]$ specifies not only the actions of B, but also how B reacts on actions in which it does not participate.

Actions of a B are first of all its ordinary actions. A Δ it executes is also its own action (it indicates its successful termination), but also a reaction on an external action guarded by B (issuing of permission for such an action). Besides, $C[\![B]\!]$ might have to comprise auxiliary actions modelling various other kinds of reactions of B on external actions.

In our case, a $C[\![B]\!]$ needs two kinds of auxiliary actions:

– In every state, there must be an action reflecting execution of an external action with no influence on B, i.e. not changing its state. We define that such an action occurs on a special gate N, for it models non-action.

– In every state, there must be an action reflecting deletion of B upon an external action, i.e. transforming B into an inactive process. We define that such an action occurs on a special gate D.

Hence in the following, G denotes N, D or an L. If a $C[\![B]\!]$ is not further combined, its auxiliary actions must be suppressed by an additional constraint: "$C[\![B]\!]|[N, D]|\mathbf{stop}$" executes only the non-auxiliary actions of B.

To become encodable in E-LOTOS with enhanced renaming, the semantics of processes $C[\![B]\!]$ needs yet another extension. Every action of a $C[\![B]\!]$ must carry, suitably encoded, sufficient information on the current state of B, i.e. on its past events and its possible future behaviours. In our case, the parameter must list identifiers I_L of all gates L on which the process is in the particular state ready to execute an action. The parameter will be needed for specification of action priorities. If a $C[\![B]\!]$ is not further combined, the parameter of its actions must be removed by a renaming operator. Let all I_G be of a type "id", while type "ids" denotes a set of identifiers I_L. In our examples, every I_G will be simply G.

Let A denote N or D. Hence the desired semantics of $C[\![\mathbf{stop}]\!]$ is

$C[\![\mathbf{stop}]\!] \xrightarrow{A(\{\})} C[\![\mathbf{stop}]\!]$

The desired semantics of $C[\![\mathbf{null}]\!]$ is

$C[\![\mathbf{null}]\!] \xrightarrow{\Delta(\{I_\Delta\})} C[\![\mathbf{null}]\!]$

$C[\![\mathbf{null}]\!] \xrightarrow{N(\{I_\Delta\})} C[\![\mathbf{null}]\!]$

$C[\![\mathbf{null}]\!] \xrightarrow{D(\{I_\Delta\})} C[\![\mathbf{stop}]\!]$

The desired semantics of a $C[\![O]\!]$ is

$C[\![O]\!] \xrightarrow{O(\{I_O\})} C[\![\mathbf{null}]\!]$

$C[\![O]\!] \xrightarrow{N(\{I_O\})} C[\![O]\!]$

$C[\![O]\!] \xrightarrow{D(\{I_O\})} C[\![\mathbf{stop}]\!]$

If x and y list the identifiers of the initial steps of a B_1 and a B_2, respectively, let $Next(x,y)$ list the identifiers of the initial steps of "$B_1; B_2$". The desired transitions of a $C[\![B_1; B_2]\!]$ are hence:

– $C[\![B_1]\!]$ executes an O in B_1, $C[\![B_2]\!]$ executes an N indicating that the O has no effect on B_2:

$$\frac{C[\![B_1]\!] \xrightarrow{O(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{N(y)} C[\![B_2']\!]}{C[\![B_1; B_2]\!] \xrightarrow{O(Next(x,y))} C[\![B_1'; B_2']\!]}$$

– $C[\![B_2]\!]$ executes an L in B_2, $C[\![B_1]\!]$ supports that by executing Δ in B_1:

$$\frac{C[\![B_1]\!] \xrightarrow{\Delta(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{L(y)} C[\![B_2']\!]}{C[\![B_1;B_2]\!] \xrightarrow{L(Next(x,y))} C[\![B_1';B_2']\!]}$$

- For an action in the environment of "$B_1;B_2$", $C[\![B_1]\!]$ and $C[\![B_2]\!]$ indicate, by co-operative execution of an N, that it has no effect on B_1 and B_2, or, by co-operative execution of a D, that it disables further execution of B_1 and B_2:

$$\frac{C[\![B_1]\!] \xrightarrow{A(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{A(y)} C[\![B_2']\!]}{C[\![B_1;B_2]\!] \xrightarrow{A(Next(x,y))} C[\![B_1';B_2']\!]}$$

Let $Next_\Pi(x,y,z)$ denote the same function as Π, except that wherever Π works with an L, $Next_\Pi(x,y,z)$ works with I_L. The desired transitions of a $C[\![B_1[\Pi]B_2]\!]$ are hence:

- $C[\![B_1]\!]$ executes an L in B_1, $C[\![B_2]\!]$ executes a D indicating that B_2 is consequently disabled:

$$\frac{C[\![B_1]\!] \xrightarrow{L(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{D(y)} C[\![B_2']\!]}{C[\![B_1[\Pi]B_2]\!] \xrightarrow{L(V_1 \cup V_2)} C[\![B_1'[\Pi]B_2']\!]}$$

where $(V_1, V_2, \{\})$ is $Next_\Pi(x,y,\{\})$ and I_L is in V_1.

- $C[\![B_2]\!]$ executes an L in B_2, $C[\![B_1]\!]$ executes a D indicating that B_1 is consequently disabled:

$$\frac{C[\![B_1]\!] \xrightarrow{D(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{L(y)} C[\![B_2']\!]}{C[\![B_1[\Pi]B_2]\!] \xrightarrow{L(V_1 \cup V_2)} C[\![B_1'[\Pi]B_2']\!]}$$

where $(V_1, V_2, \{\})$ is $Next_\Pi(x,y,\{\})$ and I_L is in V_2.

- For an action in the environment of "$B_1[\Pi]B_2$", $C[\![B_1]\!]$ and $C[\![B_2]\!]$ indicate, by co-operative execution of an N, that it has no effect on B_1 and B_2, or, by co-operative execution of a D, that it disables further execution of B_1 and B_2:

$$\frac{C[\![B_1]\!] \xrightarrow{A(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{A(y)} C[\![B_2']\!]}{C[\![B_1[\Pi]B_2]\!] \xrightarrow{A(V_1 \cup V_2)} C[\![B_1'[\Pi]B_2']\!]}$$

where $(V_1, V_2, \{\})$ is $Next_\Pi(x,y,\{\})$.

Let $Sync(\Omega)$ denote $\{I_L | (L \in (\Omega \cup \{\Delta\}))\}$. The desired transitions of a $C[\![B_1|[\Omega|\Pi]|B_2]\!]$ are hence:

- $C[\![B_1]\!]$ executes an O in B_1 outside Ω, $C[\![B_2]\!]$ executes an N indicating that the O has no effect on B_2:

$$\frac{C[\![B_1]\!] \xrightarrow{O(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{N(y)} C[\![B_2']\!]}{C[\![B_1|[\Omega|\Pi]|B_2]\!] \xrightarrow{O(V_1 \cup V_2 \cup V_3)} C[\![B_1'|[\Omega|\Pi]|B_2']\!]}$$

where (V_1, V_2, V_3) is $Next_\Pi(x,y,Sync(\Omega))$ and I_O is in V_1.

- $C[\![B_2]\!]$ executes an O in B_2 outside Ω, $C[\![B_1]\!]$ executes an N indicating that the O has no effect on B_1:

$$\frac{C[\![B_1]\!] \xrightarrow{N(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{O(y)} C[\![B_2']\!]}{C[\![B_1|[\Omega|\Pi]|B_2]\!] \xrightarrow{O(V_1 \cup V_2 \cup V_3)} C[\![B_1'|[\Omega|\Pi]|B_2']\!]}$$

where (V_1, V_2, V_3) is $Next_\Pi(x,y,Sync(\Omega))$ and I_O is in V_2.

- $C[\![B_1]\!]$ and $C[\![B_2]\!]$ co-operatively execute an L in $(\Omega \cup \{\Delta\})$, i.e. an L in B_1 and an L in B_2:

$$\frac{C[\![B_1]\!] \xrightarrow{L(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{L(y)} C[\![B_2']\!]}{C[\![B_1|[\Omega|\Pi]|B_2]\!] \xrightarrow{L(V_1 \cup V_2 \cup V_3)} C[\![B_1'|[\Omega|\Pi]|B_2']\!]}$$

where (V_1, V_2, V_3) is $Next_\Pi(x,y,Sync(\Omega))$ and I_O is in V_3.

- For an action in the environment of "$B_1|[\Omega|\Pi]|B_2$", $C[\![B_1]\!]$ and $C[\![B_2]\!]$ indicate, by co-operative execution of an N, that it has no effect on B_1 and B_2, or, by co-operative execution of a D, that it disables further execution of B_1 and B_2:

$$\frac{C[\![B_1]\!] \xrightarrow{A(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{A(y)} C[\![B_2']\!]}{C[\![B_1|[\Omega|\Pi]|B_2]\!] \xrightarrow{A(V_1 \cup V_2 \cup V_3)} C[\![B_1'|[\Omega|\Pi]|B_2']\!]}$$

where (V_1, V_2, V_3) is $Next_\Pi(x,y,Sync(\Omega))$.

The desired transitions of a $C[\![B_1[\Pi > B_2]\!]$ are:

- $C[\![B_1]\!]$ executes an O in B_1, $C[\![B_2]\!]$ executes an N indicating that the O has no effect on B_2:

$$\frac{C[\![B_1]\!] \xrightarrow{O(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{N(y)} C[\![B_2']\!]}{C[\![B_1[\Pi > B_2]\!] \xrightarrow{O(V_1 \cup V_2)} C[\![B_1'[\Pi > B_2']\!]}$$

where $(V_1, V_2, \{\})$ is $Next_\Pi(x,y,\{\})$ and I_O is in V_1.

- $C[\![B_1]\!]$ executes a Δ in B_1, $C[\![B_2]\!]$ executes a D indicating that B_2 is consequently disabled:

$$\frac{C[\![B_1]\!] \xrightarrow{\Delta(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{D(y)} C[\![B_2']\!]}{C[\![B_1[\Pi > B_2]\!] \xrightarrow{\Delta(V_1 \cup V_2)} C[\![B_1'[\Pi > B_2']\!]}$$

where $(V_1, V_2, \{\})$ is $Next_\Pi(x,y,\{\})$ and I_Δ is in V_1.

- $C[\![B_2]\!]$ executes an L in B_2, $C[\![B_1]\!]$ executes a D indicating that B_1 is consequently disabled:

$$\frac{C[\![B_1]\!] \xrightarrow{D(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{L(y)} C[\![B_2']\!]}{C[\![B_1[\Pi > B_2]\!] \xrightarrow{L(V_1 \cup V_2)} C[\![B_1'[\Pi > B_2']\!]}$$

where $(V_1, V_2, \{\})$ is $Next_\Pi(x,y,\{\})$ and I_L is in V_2.

– For an action in the environment of "$B_1[\Pi > B_2$", $C[\![B_1]\!]$ and $C[\![B_2]\!]$ indicate, by co-operative execution of an N, that it has no effect on B_1 and B_2, or, by co-operative execution of a D, that it disables further execution of B_1 and B_2:

$$\frac{C[\![B_1]\!] \xrightarrow{A(x)} C[\![B_1']\!], C[\![B_2]\!] \xrightarrow{A(y)} C[\![B_2']\!]}{C[\![B_1[\Pi > B_2]\!] \xrightarrow{A(V_1 \cup V_2)} C[\![B_1'[\Pi > B_2']\!]}$$

where $(V_1, V_2, \{\})$ is $Next_\Pi(x, y, \{\})$.

4.2.3. *Encoding in E-LOTOS with enhanced renaming*

We begin by observing that functions $Next(x, y)$ and $Next_\Pi(x, y, z)$ can be easily encoded in E-LOTOS.

An E-LOTOS encoding of $C[\![\textbf{stop}]\!]$ is
"**loop** $N!\{\}[]D!\{\}$ **endloop**".

Assuming that Δ denotes a regular E-LOTOS gate name, an encoding for $C[\![\textbf{null}]\!]$ would be
"(**loop** $\Delta!\{I_\Delta\}[]N!\{I_\Delta\}$ **endloop**
$[> (D!\{I_\Delta\}; C[\![\textbf{stop}]\!]))$".

An encoding for $C[\![O]\!]$ is
"(**loop** $N!\{I_O\}$ **endloop**
$[> ((O!\{I_O\}; C[\![\textbf{null}]\!])[](D!\{I_O\}; C[\![\textbf{stop}]\!])))$".

Encodings for $C[\![B_1; B_2]\!]$, $C[\![B_1[\Pi]B_2]\!]$, $C[\![B_1|[\Omega|\Pi]|B_2]\!]$ and $C[\![B_1[\Pi > B_2]\!]$ are given in Fig. 1, 3, 5 and 7, respectively, where $\mathcal{G}(B)$ denotes the visible gates of a B, and $\mathcal{I}(B)$ the identifiers I_L of L in $\mathcal{G}(B)$. Every such $C[\![B_1 * B_2]\!]$ (in prefix notation $C[\![*(B_1, B_2)]\!]$) is structured according to the following strategy for encoding a $C[\![*(B_1, \ldots, B_n)]\!]$ (see the examples in Fig. 2, 4, 6 and 8):

– For all B_i: Every action $G_i(x_i)$ of $C[\![B_i]\!]$ is split into all such $G(I_{G_1}, x_1, \ldots, I_{G_n}, x_n)$ that $C[\![*(B_1, \ldots, B_n)]\!]$ could, according to the semantics of "$*$", have an action $G(x)$ resulting from co-operative execution of $G_1(x_1)$ in $C[\![B_1]\!]$, $G_2(x_2)$ in $C[\![B_2]\!]$,... and $G_n(x_n)$ in $C[\![B_n]\!]$. For every $G_i(x_i)$, one also specifies a dummy new name $G_i(i)$ (not intended for execution), to secure that there is at least one new name per action, for otherwise it would be executable in its original form. The process obtained from $C[\![B_i]\!]$ by the described action renamings is a $C_i[\![*(B_1, \ldots, B_n)]\!]$.
– All the constructed processes $C_i[\![*(B_1, \ldots, B_n)]\!]$ are put in parallel composition with total synchronization. Every action of the composite process $C'[\![*(B_1, \ldots, B_n)]\!]$ is a $G(I_{G_1}, x_1, \ldots, I_{G_n}, x_n)$ resulting from a $G(I_{G_1}, x_1, \ldots, I_{G_n}, x_n)$ in $C_1[\![*(B_1, \ldots, B_n)]\!]$, a $G(I_{G_1}, x_1, \ldots, I_{G_n}, x_n)$ in $C_2[\![*(B_1, \ldots, B_n)]\!]$,... and a $G(I_{G_1}, x_1, \ldots, I_{G_n}, x_n)$ in $C_n[\![*(B_1, \ldots, B_n)]\!]$, i.e. from a $G_1(x_1)$ in $C[\![B_1]\!]$, a $G_2(x_2)$ in $C[\![B_2]\!]$,... and a $G_n(x_n)$ in $C[\![B_n]\!]$, as required. Any action $G_i(i)$ of a $C_i[\![*(B_1, \ldots, B_n)]\!]$ is non-executable within the context, as intended.
– Every action $G(I_{G_1}, x_1, \ldots, I_{G_n}, x_n)$ of $C'[\![*(B_1, \ldots, B_n)]\!]$ is renamed into the $G(x)$ which it denotes in $C[\![*(B_1, \ldots, B_n)]\!]$.

Such encoding directly reflects the dynamic semantics of $C[\![*(B_1, \ldots, B_n)]\!]$.

4.3. *Dealing with data and time*

In the language we implemented in Section 4.2, all actions were untimed and carried no data. In this section, we briefly discuss how to overcome the restrictions.

Data handling can be introduced as follows:

– When designing a language which is to be encoded in E-LOTOS, let processes B maintain and update data variables and let actions L carry data, as desired.
– When conceiving the implementation semantics, include, as convenient, in the action parameters representing the current state of a process B information on the data stored by B or potentially carried by its future actions. In such a way, it will be possible to use the data in predicates defining which are the actions that synchronize when the processes $C[\![B]\!]$ to which they belong are combined.

As an example, suppose that we are implementing a "$B_1; B_2$" specified as "$O_1?x; O_2?y[y < x]$". Obviously, x must be carried by Δ actions of B_1, and the legal combinations of a Δ in $C[\![B_1]\!]$ and an O_2 in $C[\![B_2]\!]$ are those where the y carried by O_2 is smaller than the x carried by Δ.

Timed actions can be introduced as follows:

– When designing a language which is to be encoded in E-LOTOS, design it as an untimed language, but let every action L carry a parameter

```
$C[\![B_1;B_2]\!] :=$ rename forall $G \in \mathcal{G}(C'[\![B_1;B_2]\!])$ : action $G(\mathrm{id}, ?x : \mathrm{ids}, \mathrm{id}, ?y : \mathrm{ids})$ is $G!Next(x,y)$ endfor
        in $C'[\![B_1;B_2]\!]$ endren
$C'[\![B_1;B_2]\!] := C_1[\![B_1;B_2]\!] || C_2[\![B_1;B_2]\!]$
$C_1[\![B_1;B_2]\!] :=$ rename forall $O \in \mathcal{G}(C[\![B_1]\!])$ : action $O(?x : \mathrm{ids})$ is $O(!I_O, !x, !I_N, ?y : \mathrm{ids})[\mathrm{issubset}(y, \mathcal{I}(C[\![B_2]\!]))]$ endfor
                if $\Delta \in \mathcal{G}(C[\![B_1]\!])$ then forall $L \in \mathcal{G}(C[\![B_2]\!])$ : action $\Delta(?x : \mathrm{ids})$ is $L(!I_\Delta, !x, !I_L, ?y : \mathrm{ids})$
                        $[\mathrm{issubset}(y, \mathcal{I}(C[\![B_2]\!]))$ and $\mathrm{isin}(I_L, y)]$ endfor endthen
                forall $A \in \{N, D\}$ : action $A(?x : \mathrm{ids})$ is $A(!I_A, !x, !I_A, ?y : \mathrm{ids})[\mathrm{issubset}(y, \mathcal{I}(C[\![B_2]\!]))]$ endfor
                forall $G \in \mathcal{G}(C[\![B_1]\!])$ : action $G(\mathrm{ids})$ is $G!1$ endfor
        in $C[\![B_1]\!]$ endren
$C_2[\![B_1;B_2]\!] :=$ rename forall $L \in \mathcal{G}(C[\![B_2]\!])$ : action $L(?y : \mathrm{ids})$ is $O(!I_\Delta, ?x : \mathrm{ids}, !I_L, !y)$
                        $[\mathrm{issubset}(x, \mathcal{I}(C[\![B_1]\!]))$ and $\mathrm{isin}(I_\Delta, x)]$ endfor
                forall $O \in \mathcal{G}(C[\![B_1]\!])$ : action $N(?y : \mathrm{ids})$ is $O(!I_O, ?x : \mathrm{ids}, !I_N, !y)$
                        $[\mathrm{issubset}(x, \mathcal{I}(C[\![B_1]\!]))$ and $\mathrm{isin}(I_O, x)]$ endfor
                forall $A \in \{N, D\}$ : action $A(?y : \mathrm{ids})$ is $A(!I_A, ?x : \mathrm{ids}, !I_A, !y)[\mathrm{issubset}(x, \mathcal{I}(C[\![B_1]\!]))]$ endfor
                forall $G \in \mathcal{G}(C[\![B_2]\!])$ : action $G(\mathrm{ids})$ is $G!2$ endfor
        in $C[\![B_2]\!]$ endren
```

Fig. 1. Encoding of $C[\![B_1;B_2]\!]$ in E-LOTOS with enhanced renaming.

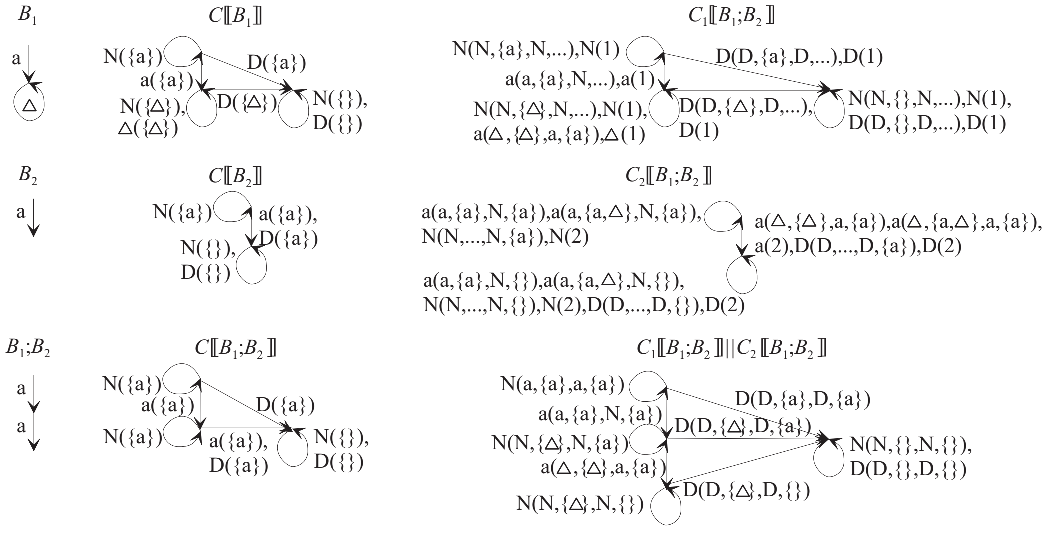

Fig. 2. Example implementation of sequential composition.

supposingly representing the absolute execution time of L, and restrict its value as desired.

– When implementing a specified process B, introduce an additional constraint representing a global clock. The clock process will synchronize with every L in $C[\![B]\!]$ and take care that its execution-time parameter equals the current time.

As an example, take the process "$O_1; O_2@!3$", executing O_2 3 time units after O_1. Introducing absolute-execution-time parameters, we rewrite the process into

"$O_1?x : \mathrm{time}; O_2?y : \mathrm{time}[y = x + 3]$".

An adequate clock will allow $O_1(x)$ only at time x, and $O_2(y)$ only at time y.

5. Discussion and conclusions

Among the LOTOS (or E-LOTOS) specification styles [11], the constraint-oriented style yields specifications which are the most difficult to implement in an efficient way. However, when one wants to specify a really complicated behaviour, the style is usually the best choice, for it facilitates abstract and declarative specification.

```
$C[\![B_1[\Pi]B_2]\!] :=$ rename
        forall $G \in \mathcal{G}(C'[\![B_1[\Pi]B_2]\!])$ : action $G(\mathrm{id}, ?x : \mathrm{ids}, \mathrm{id}, ?y : \mathrm{ids})$ is $G?z : \mathrm{ids}$
         [var $V_1 : \mathrm{ids}, V_2 : \mathrm{ids}$ in $(?V_1, ?V_2,$ any $: \mathrm{ids}) := Next_\Pi(x, y, \{\}); z = \mathrm{union}(V_1, V_2)$ endvar] endfor
        in $C'[\![B_1[\Pi]B_2]\!]$ endren
$C'[\![B_1[\Pi]B_2]\!] := C_1[\![B_1[\Pi]B_2]\!] || C_2[\![B_1[\Pi]B_2]\!]$
$C_1[\![B_1[\Pi]B_2]\!] :=$ rename
        forall $L \in \mathcal{G}(C[\![B_1]\!])$ : action $L(?x : \mathrm{ids})$ is $L(!I_L, !x, !I_D, ?y : \mathrm{ids})[\mathrm{issubset}(y, \mathcal{I}(C[\![B_2]\!]))$ and
          var $V_1 : \mathrm{ids}$ in $(?V_1,$ any $: \mathrm{ids},$ any $: \mathrm{ids}) := Next_\Pi(x, y, \{\}); \mathrm{isin}(I_L, V_1)$ endvar] endfor
        forall $L \in \mathcal{G}(C[\![B_2]\!])$ : action $D(?x : \mathrm{ids})$ is $L(!I_D, !x, !I_L, ?y : \mathrm{ids})[\mathrm{issubset}(y, \mathcal{I}(C[\![B_2]\!]))$ and
          var $V_2 : \mathrm{ids}$ in (any $: \mathrm{ids}, ?V_2,$ any $: \mathrm{ids}) := Next_\Pi(x, y, \{\}); \mathrm{isin}(I_L, V_2)$ endvar] endfor
        forall $A \in \{N, D\}$ : action $A(?x : \mathrm{ids})$ is $A(!I_A, !x, !I_A, ?y : \mathrm{ids})[\mathrm{issubset}(y, \mathcal{I}(C[\![B_2]\!]))]$ endfor
        forall $G \in \mathcal{G}(C[\![B_1]\!])$ : action $G(\mathrm{ids})$ is $G!1$ endfor
        in $C[\![B_1]\!]$ endren
$C_2[\![B_1[\Pi]B_2]\!] :=$ rename
        forall $L \in \mathcal{G}(C[\![B_2]\!])$ : action $L(?y : \mathrm{ids})$ is $L(!I_D, ?x : \mathrm{ids}, !I_L, !y)[\mathrm{issubset}(x, \mathcal{I}(C[\![B_1]\!]))$ and
          var $V_2 : \mathrm{ids}$ in (any $: \mathrm{ids}, ?V_2,$ any $: \mathrm{ids}) := Next_\Pi(x, y, \{\}); \mathrm{isin}(I_L, V_2)$ endvar] endfor
        forall $L \in \mathcal{G}(C[\![B_1]\!])$ : action $D(?y : \mathrm{ids})$ is $L(!I_L, ?x : \mathrm{ids}, !I_D, !y)[\mathrm{issubset}(x, \mathcal{I}(C[\![B_1]\!]))$ and
          var $V_1 : \mathrm{ids}$ in $(?V_1,$ any $: \mathrm{ids},$ any $: \mathrm{ids}) := Next_\Pi(x, y, \{\}); \mathrm{isin}(I_L, V_1)$ endvar] endfor
        forall $A \in \{N, D\}$ : action $A(?y : \mathrm{ids})$ is $A(!I_A, ?x : \mathrm{ids}, !I_A, !y)[\mathrm{issubset}(x, \mathcal{I}(C[\![B_1]\!]))]$ endfor
        forall $G \in \mathcal{G}(C[\![B_2]\!])$ : action $G(\mathrm{ids})$ is $G!2$ endfor
        in $C[\![B_2]\!]$ endren
```

Fig. 3. Encoding of $C[\![B_1[\Pi]B_2]\!]$ in E-LOTOS with enhanced renaming.

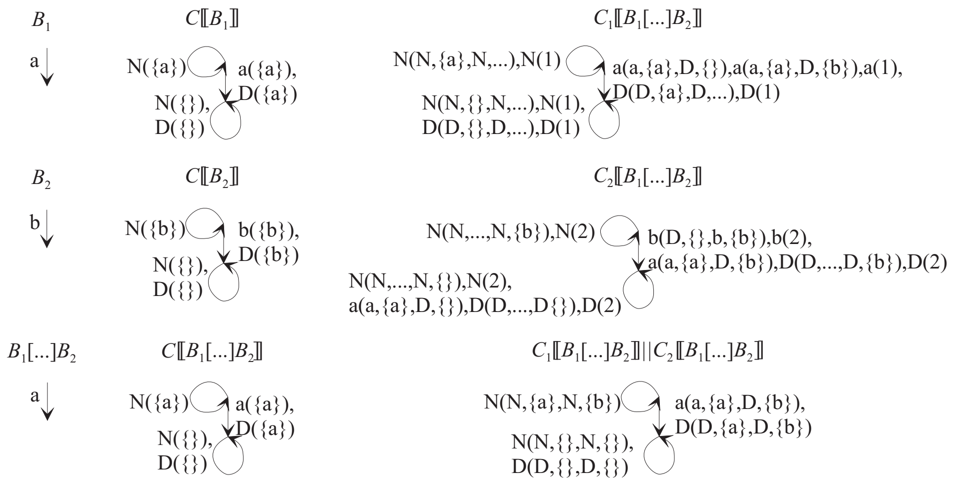

Fig. 4. Example implementation of choice, where a has a higher priority than b.

In the paper, we sketched a method for constraint-oriented specification of new process-composition operators. It consists of two steps. In the first step, one designs a process-algebraic specification language best meeting one's specific needs. If the language introduces only events which are fully controllable and freely renamable (for every event, one can at least pretend that it is), the dynamic semantics of the language can then be mechanically rewritten into a form which allows mechanic translation of process specifications into E-LOTOS. If supported by a tool, the method would help those users of E-LOTOS (or its enhanced successors) for whom the constraint-oriented style is too abstract, but would still want to specify non-standard forms of process composition.

Applying the above method, we observed that it strongly builds upon action splitting. In E-LOTOS, an action can be split only with respect

```
$C\llbracket B_1|[\Omega|\Pi]|B_2\rrbracket$ := rename forall $G \in \mathcal{G}(C'\llbracket B_1|[\Omega|\Pi]|B_2\rrbracket)$ : action $G(\mathrm{id}, ?x : \mathrm{ids}, \mathrm{id}, ?y : \mathrm{ids})$ is $G?z : \mathrm{ids}$
                    [var $V_1 : \mathrm{ids}, V_2 : \mathrm{ids}, V_3 : \mathrm{ids}$ in $(?V_1, ?V_2, ?V_3) := Next_\Pi(x, y, Sync(\Omega))$;
                         $z = \mathrm{union}(\mathrm{union}(V_1, V_2), V_3)$ endvar] endfor
          in $C'\llbracket B_1|[\Omega|\Pi]|B_2\rrbracket$ endren
$C'\llbracket B_1|[\Omega|\Pi]|B_2\rrbracket := C_1\llbracket B_1|[\Omega|\Pi]|B_2\rrbracket || C_2\llbracket B_1|[\Omega|\Pi]|B_2\rrbracket$
$C_1\llbracket B_1|[\Omega|\Pi]|B_2\rrbracket$ := rename
          forall $L \in ((\Omega \cup \{\Delta\}) \cap \mathcal{G}(C\llbracket B_1\rrbracket) \cap \mathcal{G}(C\llbracket B_2\rrbracket))$ : action $L(?x : \mathrm{ids})$ is $L(!I_L, !x, !I_L, ?y : \mathrm{ids})$
            [issubset($y, \mathcal{I}(C\llbracket B_2\rrbracket)$) and
            var $V_3$ : ids in (any : ids, any : ids, $?V_3$) := $Next_\Pi(x, y, Sync(\Omega))$; isin($I_L, V_3$) endvar] endfor
          forall $O \in (\mathcal{G}(C\llbracket B_1\rrbracket)\backslash\Omega)$ : action $O(?x : \mathrm{ids})$ is $O(!I_O, !x, !I_N, ?y : \mathrm{ids})$[issubset($y, \mathcal{I}(C\llbracket B_2\rrbracket)$) and
            var $V_1$ : ids in ($?V_1$, any : ids, any : ids) := $Next_\Pi(x, y, Sync(\Omega))$; isin($I_O, V_1$) endvar] endfor
          forall $O \in (\mathcal{G}(C\llbracket B_2\rrbracket)\backslash\Omega)$ : action $N(?x : \mathrm{ids})$ is $O(!I_N, !x, !I_O, ?y : \mathrm{ids})$[issubset($y, \mathcal{I}(C\llbracket B_2\rrbracket)$) and
            var $V_2$ : ids in (any : ids, $?V_2$, any : ids) := $Next_\Pi(x, y, Sync(\Omega))$; isin($I_O, V_2$) endvar] endfor
          forall $A \in \{N, D\}$ : action $A(?x : \mathrm{ids})$ is $A(!I_A, !x, !I_A, ?y : \mathrm{ids})$[issubset($y, \mathcal{I}(C\llbracket B_2\rrbracket)$)] endfor
          forall $G \in \mathcal{G}(C\llbracket B_1\rrbracket)$ : action $G(\mathrm{ids})$ is $G!1$ endfor
          in $C\llbracket B_1\rrbracket$ endren
$C_2\llbracket B_1|[\Omega|\Pi]|B_2\rrbracket$ := rename
          forall $L \in ((\Omega \cup \{\Delta\}) \cap \mathcal{G}(C\llbracket B_1\rrbracket) \cap \mathcal{G}(C\llbracket B_2\rrbracket))$ : action $L(?y : \mathrm{ids})$ is $L(!I_L, ?x : \mathrm{ids}, !I_L, !y)$
            [issubset($x, \mathcal{I}(C\llbracket B_1\rrbracket)$) and
            var $V_3$ : ids in (any : ids, any : ids, $?V_3$) := $Next_\Pi(x, y, Sync(\Omega))$; isin($I_L, V_3$) endvar] endfor
          forall $O \in (\mathcal{G}(C\llbracket B_2\rrbracket)\backslash\Omega)$ : action $O(?y : \mathrm{ids})$ is $O(!I_N, ?x : \mathrm{ids}, !I_O, !y)$[issubset($x, \mathcal{I}(C\llbracket B_1\rrbracket)$) and
            var $V_2$ : ids in (any : ids, $?V_2$, any : ids) := $Next_\Pi(x, y, Sync(\Omega))$; isin($I_O, V_2$) endvar] endfor
          forall $O \in (\mathcal{G}(C\llbracket B_1\rrbracket)\backslash\Omega)$ : action $N(?y : \mathrm{ids})$ is $O(!I_O, ?x : \mathrm{ids}, !I_N, !y)$[issubset($x, \mathcal{I}(C\llbracket B_1\rrbracket)$) and
            var $V_1$ : ids in ($?V_1$, any : ids, any : ids) := $Next_\Pi(x, y, Sync(\Omega))$; isin($I_O, V_1$) endvar] endfor
          forall $A \in \{N, D\}$ : action $A(?y : \mathrm{ids})$ is $A(!I_A, ?x : \mathrm{ids}, !I_A, !y)$[issubset($x, \mathcal{I}(C\llbracket B_1\rrbracket)$)] endfor
          forall $G \in \mathcal{G}(C\llbracket B_2\rrbracket)$ : action $G(\mathrm{ids})$ is $G!2$ endfor
          in $C\llbracket B_2\rrbracket$ endren
```

Fig. 5. Encoding of $C\llbracket B_1|[\Omega|\Pi]|B_2\rrbracket$ in E-LOTOS with enhanced renaming.

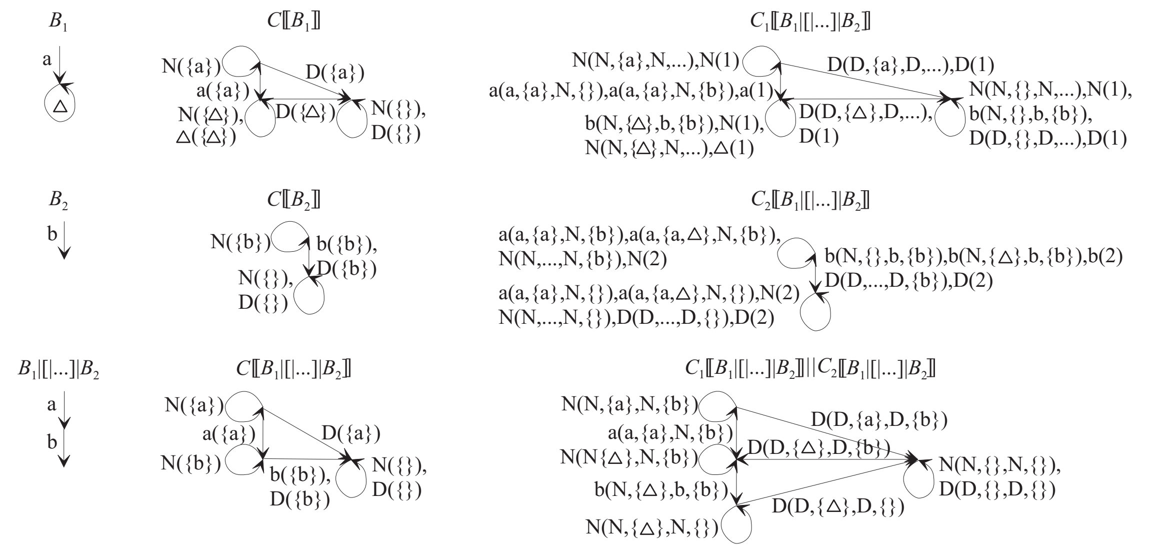

Fig. 6. Example implementation of parallel composition, where a has a higher priority than b.

to a parameter introduced specifically for the purpose. Having found that inconvenient, we decided to propose a generalization of the E-LOTOS renaming operator. Besides being more powerful, the new renaming operator also has a more natural and non-restrictive semantics.

We conclude by summarizing that the proposed generalization would help in two ways:

```
$C[\![B_1[\Pi > B_2]]\!] :=$ rename forall $G \in \mathcal{G}(C'[\![B_1[\Pi > B_2]]\!])$ : action $G(\text{id}, ?x : \text{ids}, \text{id}, ?y : \text{ids})$ is $G?z : \text{ids}$
                          [var $V_1 : \text{ids}, V_2 : \text{ids}$ in $(?V_1, ?V_2, \textbf{any} : \text{ids}) := Next_\Pi(x, y, \{\}); z = \text{union}(V_1, V_2)$ endvar]
                          endfor
                    in $C'[\![B_1[\Pi > B_2]]\!]$ endren
$C'[\![B_1[\Pi > B_2]]\!] := C_1[\![B_1[\Pi > B_2]]\!] || C_2[\![B_1[\Pi > B_2]]\!]$
$C_1[\![B_1[\Pi > B_2]]\!] :=$ rename
                    forall $O \in \mathcal{G}(C[\![B_1]\!])$ : action $O(?x : \text{ids})$ is $O(!I_O, !x, !I_N, ?y : \text{ids})[\text{issubset}(y, \mathcal{I}(C[\![B_2]\!]))$ and
                      var $V_1$ : ids in $(?V_1, \textbf{any} : \text{ids}, \textbf{any} : \text{ids}) := Next_\Pi(x, y, \{\}); \text{isin}(I_O, V_1)$ endvar] endfor
                    if $\Delta \in \mathcal{G}(C[\![B_1]\!])$ then action $\Delta(?x : \text{ids})$ is $\Delta(!I_\Delta, !x, !I_D, ?y : \text{ids})[\text{issubset}(y, \mathcal{I}(C[\![B_2]\!]))$ and
                      var $V_1$ : ids in $(?V_1, \textbf{any} : \text{ids}, \textbf{any} : \text{ids}) := Next_\Pi(x, y, \{\}); \text{isin}(I_\Delta, V_1)$ endvar] endif
                    forall $L \in \mathcal{G}(C[\![B_2]\!])$ : action $D(?x : \text{ids})$ is $L(!I_D, !x, !I_L, ?y : \text{ids})[\text{issubset}(y, \mathcal{G}(C[\![B_2]\!]))$ and
                      var $V_2$ : ids in $(\textbf{any} : \text{ids}, ?V_2, \textbf{any} : \text{ids}) := Next_\Pi(x, y, \{\}); \text{isin}(I_L, V_2)$ endvar] endfor
                    forall $A \in \{N, D\}$ : action $A(?x : \text{ids})$ is $A(!I_A, !x, !I_A, ?y : \text{ids})[\text{issubset}(y, \mathcal{I}(C[\![B_2]\!]))]$ endfor
                    forall $G \in \mathcal{G}(C[\![B_1]\!])$ : action $G(\text{ids})$ is $G!1$ endfor
                    in $C[\![B_1]\!]$ endren
$C_2[\![B_1[\Pi > B_2]]\!] :=$ rename
                    forall $L \in \mathcal{G}(C[\![B_2]\!])$ : action $L(?y : \text{ids})$ is $L(!I_D, ?x : \text{ids}, !I_L, !y)[\text{issubset}(x, \mathcal{I}(C[\![B_1]\!]))$ and
                      var $V_2$ : ids in $(\textbf{any} : \text{ids}, ?V_2, \textbf{any} : \text{ids}) := Next_\Pi(x, y, \{\}); \text{isin}(I_L, V_2)$ endvar] endfor
                    forall $O \in \mathcal{G}(C[\![B_1]\!])$ : action $N(?y : \text{ids})$ is $O(!I_O, ?x : \text{ids}, !I_N, !y)[\text{issubset}(x, \mathcal{G}(C[\![B_1]\!]))$ and
                      var $V_1$ : ids in $(?V_1, \textbf{any} : \text{ids}, \textbf{any} : \text{ids}) := Next_\Pi(x, y, \{\}); \text{isin}(I_O, V_1)$ endvar] endfor
                    action $D(?y : \text{ids})$ is $\Delta(!I_\Delta, ?x : \text{ids}, !I_D, !y)[\text{issubset}(x, \mathcal{G}(C[\![B_1]\!]))$ and
                      var $V_1$ : ids in $(?V_1, \textbf{any} : \text{ids}, \textbf{any} : \text{ids}) := Next_\Pi(x, y, \{\}); \text{isin}(I_\Delta, V_1)$ endvar]
                    forall $A \in \{N, D\}$ : action $A(?y : \text{ids})$ is $A(!I_A, ?x : \text{ids}, !I_A, !y)[\text{issubset}(x, \mathcal{I}(C[\![B_1]\!]))]$ endfor
                    forall $G \in \mathcal{G}(C[\![B_2]\!])$ : action $G(\text{ids})$ is $G!2$ endfor
                    in $C[\![B_2]\!]$ endren
```

Fig. 7. Encoding of $C[\![B_1[\Pi > B_2]]\!]$ in E-LOTOS with enhanced renaming.

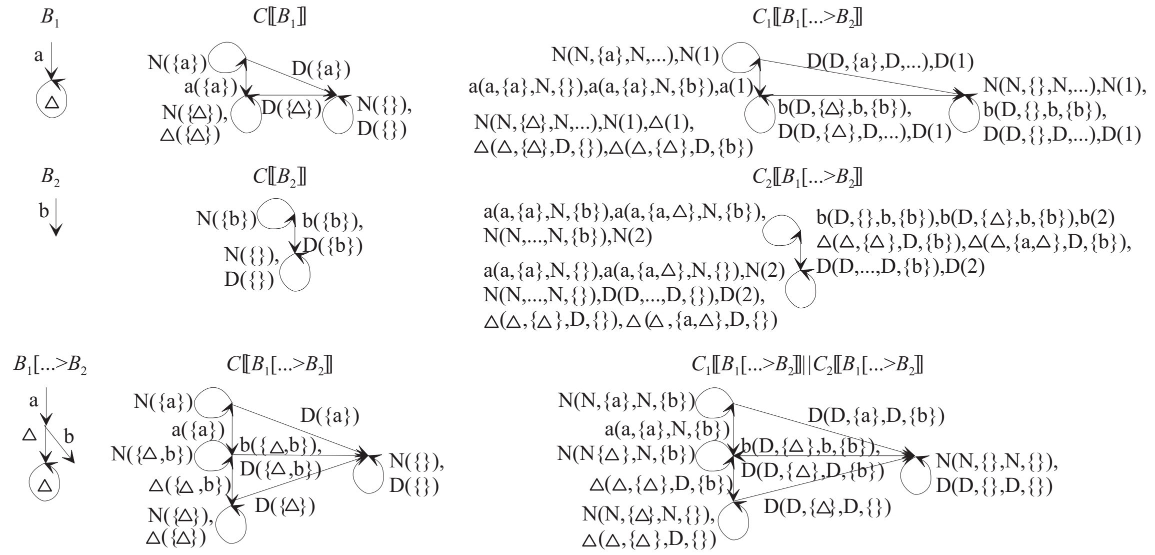

Fig. 8. Example implementation of disabling, where a has a higher priority then b.

– It would make the renaming operator much more easy to understand and convenient to use, supporting further development of E-LOTOS into a “second-generation” formal description technique [3].

– The enhanced operator would simplify specification of non-standard forms of process composition and thereby construction of compilers from custom-designed process-algebraic specification languages to E-LOTOS.

References

[1] T. Bolognesi and E. Brinksma, Introduction to the ISO specification language LOTOS, *Computer Networks and ISDN Systems* **14** (1987) 25–59.

[2] H. Garavel, A wish list for the behaviour part of E-LOTOS, Input document [LG5] to the ISO/IEC JTC1/SC21/WG7 Meeting on Enhancements to LOTOS, Liège, December 1995.

[3] H. Garavel and M. Sighireanu, Towards a second generation of formal description techniques - rationale for the design of E-LOTOS, in: J.-F. Groote, B. Luttik, and J. van Wamel (eds.), *Proc. of the 3rd International Workshop on Formal Methods for Industrial Critical Systems FMICS'98*, Amsterdam, The Netherlands, May 1998, 187–230.

[4] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice-Hall International, 1985.

[5] ISO, *LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, ISO 8807, ISO – Information Processing Systems – Open Systems Interconnection, 1989.

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

[7] M. Kapus-Kolar, Specifying broadcast communication in a sublanguage of E-LOTOS, in: B. Zajc and M. Tkalčič (eds.), *Proc. of the IEEE EUROCON'2003: Computer as a Tool*, Ljubljana, September 2003, vol.II, pp. 2–6.

[8] M. Kapus-Kolar, Specifying action priorities in a sublanguage of E-LOTOS, in: D. Begušič, N. Rožič (eds.), *Proc. of the 11th International Conference on Software, Telecommunications & Computer Networks SoftCOM'2003*, Split, October 2003, pp. 247–251.

[9] M. Kapus-Kolar, Specifying late decisions in a subset of E-LOTOS, submitted for publication, 2003.

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

[11] C. A. Vissers, G. Scollo, M. van Sinderen, and H. Brinksma, Specification styles in distributed systems design and verification, *Theoretical Computer Science* **89** (1991) 179–206.

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, Ljubljana, where she is currently a researcher at the Department of Digital Communications and Networks. Her current research interests include formal specification techniques and methods for development of distributed systems and computer networks.