AFNOR Comments Accompanying Vote of Disapproval on the Committee Draft ISO/IEC JTC1/SC21/WG7 "Enhancements to LOTOS" (1.21.20.2.3)

June 1997

AFNOR votes AGAINST promoting the Committee Draft on Enhancements to LOTOS (Date: January 1997) as a Final Committee Draft.

The reasons for this NO vote are twofold:

- First, the current Committee Draft is *not complete* nor *mature*. Several important issues in the definition of E-LOTOS are left open. For instance, the definition of the predefined library of types and processes is not provided, although having predefined types was a design goal for E-LOTOS. Also, the dynamic semantics has not been proven to be consistent.
- Second, some technical design decisions in the Committee Draft are *not appropriate*. AFNOR expresses strong concerns about the proposed definition of E-LOTOS, which results in a language that is:
  - overly complex
  - non-intuitive for the users
  - unsafe (e.g., type-checking at run-time)
  - difficult to implement
  - not interoperable and not compatible with other languages standardized by ISO/IEC (including IDL).

In that sense, the initial goals set for enhancing LOTOS have not been reached yet by the current Committee Draft.

As it is now, the proposed Committee Draft can not replace the existing International Standard IS 8807.

To progress the work, AFNOR makes a number of Technical Comments, which should be taken into account for issuing a revised Committee Draft.

These Technical Comments primarily aim at reducing the complexity of E-LOTOS, in order to have a smaller and simpler language.

In the sequel, the term "CD" will refer to the Committee Draft on Enhancements to LOTOS (1.21.20.2.3) dated January 1997; the term "revised CD" will refer to a next, revised Committee Draft, to be issued (probably after the Helsinki WG7 meeting).

ITEM: **AFNOR-1** **Abbreviating gate parameter lists**

CLAUSE: General

QUALIFIER: Major Technical

RATIONALE:

A frequent complaint about standard LOTOS concerns the lists of gates used in process instantiations. Such gate lists are tedious and error prone.

On several industrial, large-size specifications (3,000 lines and more), one noticed that **20%** of the LOTOS code was devoted exclusively to these gate parameter lists.

A proposal for having a "..." notation to abbreviate gate lists has been circulating since 1994. The rationale for this proposal are in [Gar94b]. This proposal was present in the output document of the E-LOTOS interim meeting in Liège (December 1995) and a finalized proposal is given the output document of the Kansas City meeting (May 1996).

However, it appears that the idea was lost, since the CD does not address this problem.

PROPOSED CHANGE:

The revised CD should take the problem into account and reuse the solution already available in the output document of the Kansas City meeting. It is worth noticing that this proposal is only a matter of static semantics and does not affect the dynamic semantics.

ITEM: **AFNOR-2** **Mixing "in" and "out" parameters freely**

CLAUSE: General

QUALIFIER: Major Technical

RATIONALE:

The CD does not allow "**in**" and "**out**" parameters (for functions and processes) to be combined freely: all "**in**" parameters have to come first, followed by all the "**out**" (clause (i-body4) p. 54 for processes and clause (i-body6) for functions).

This syntactic restriction is unsuitable: standard programming languages (C, C++, Pascal, Ada, etc.) as well as IDL allow "**in**" and "**out**" parameters to be mixed in free order.

Such a restriction will prevent E-LOTOS to be used as a language for specifying formally the behaviour of ODP objects whose interface is described in IDL.

It will also prevent E-LOTOS users from invoking external functions written in other languages or available in libraries (for instance, the `strcat` and `strtok` functions of the POSIX library).

PROPOSED CHANGE:

This restriction should be removed from the a revised CD. A solution is already available in the Kansas City output document.

ITEM: **AFNOR-3** **Abbreviating value parameter lists**

CLAUSE: General

QUALIFIER: Major Technical

RATIONALE:

To achieve symmetry with abbreviated gate parameter lists, it is also desirable to have abbreviated value parameter lists for processes and functions.

A proposal for having a "..." notation to abbreviate value parameter lists has been circulating since 1994. The rationale for this proposal are in [Gar94b]. This proposal was present in the output document of the E-LOTOS interim meeting in Liège (December 1995) and a finalized proposal is given the output document of the Kansas City meeting (May 1996).

For instance, under this proposal, if P is a process declared:

$$\textbf{process } P\ [G, H]\ (X, Y, Z : \texttt{Nat}) : \textbf{exit (none) is } ... \textbf{ endproc}$$

then an abbreviated instantiation of the form:

$$P\ [...](Y\ \texttt{=>}\ Y + 1, ...)$$

is syntactically equivalent to:

$$P\ [G, H](X, Y + 1, Z)$$

Besides the advantage of being more concise, the abbreviated form has also the merit of emphasizing value modifications (here, it is clear that only Y is modified).

PROPOSED CHANGE:

The revised CD should reuse the solution already available in the output document of the Kansas City meeting. It is worth noticing that this proposal is only a matter of static semantics: any abbreviated instantiation can be replaced statically by its extensive (LOTOS-like) form.

ITEM: **AFNOR-4** **"etc" clause in function/process declarations**

CLAUSE: General

QUALIFIER: Major Technical

RATIONALE:

It is not clear whether the CD allows the "**etc**" clause in function and process declarations, i.e., if the value parameter lists for processes and functions are extensible records or not.

The syntax definition (chapter 3, p. 48, clause (D3) sq.) does not allow the "**etc**" clause (extensible record) in function and process value parameter lists.

However, the "**etc**" clause is allowed, in the same context, by the corresponding semantic definitions (chapter 4, p. 58, clauses (D3)–(D6).)

This contradiction should be clarified.

We consider that "**etc**" clauses in value parameter lists should be forbidden, as they have really questionable and unsafe consequences. For instance, they enable to increase the number of value parameters in recursive process calls:

```
process P (X:nat, etc) : exit (none) is
   P (X => 1, Y => 2, Z => 3)
endproc
```

It seems reasonable to forbid such process definitions.

PROPOSED CHANGE:

The revised CD should forbid "**etc**" clauses in process and function value parameter lists.

ITEM: **AFNOR-5** **Overloading**

CLAUSE: General

QUALIFIER: Major Technical

RATIONALE:

Although overloading already exist in LOTOS, the current CD for E-LOTOS does not support overloading.

As a consequence, there cannot be at the same time an equality operator named "=" for integers and and equality operator named "=" for booleans. The same situation applies to all other operators (arithmetic, relational, etc.)

This is probably the reason why the CD provides no base environment for E-LOTOS. Not only, such a base environment does not exist at the time being, but it will never exist under the assumptions upon which the current CD is built.

On page 13, in Section 2.1.1.1, it is claimed that "*In the base language, all type, constructor, function and process identifiers must be unique – all treatment of overloading is left to the module language*".

Technically speaking, this assertion is a complete non-sense, because the treatment of overloading cannot be separated from type-checking: the type informations computed during type-checking are needed to solve overloading, and vice-versa.

Also, on page 43, Section 2.2.1.8, it is suggested to have built-in data types with a rich-term syntax.

However, the proposal for rich-term syntax [Pec94, Pec96] relies on overloading.

PROPOSED CHANGE:

Preserve overloading in E-LOTOS as it already exists in LOTOS. This will provide upward compatibly by simple translation and allow a base environment with a rich-term syntax to be defined.

The static semantics for type-checking with overloading is provided in the Kansas City output document. The proposal for rich-term syntax is in [Pec94].

ITEM: **AFNOR-6** **Restricting the "any" type to gates only**

CLAUSE: General

QUALIFIER: Major Technical

RATIONALE:

The original proposal for gate typing [Gar94a, Gar95] introduced the possibility of declaring some gates of type "**any**": such gates could accept any list of experiment offers.

Although the possibility of declaring gates of type "**any**" goes against the intent of E-LOTOS designers to have strongly-typed gates, this feature can be useful for "programming in-the-small" and for backward compatibility with LOTOS, where all gates are entirely untyped.

In the original proposal, the "**any**" feature was only available for gate types (called "channels"). Still in this proposal, the only difference between typed gate and "**any**" gates was in static semantics: for typed gates, there were additional static semantics constraints that did not apply to "**any**" gates.

This original proposal was altered in two successive steps:

1. Gate types and "standard" types have been merged into a single concept. Although this "unification" might be found elegant for defining the semantics, it is not clear that it should be made visible to the user.

   In the CD, we are in the situation where variables and gates have the same types, but are syntactically different (for instance, in process instantiations, they are put in different parameter lists).

2. The next step was to allow for variables the "**any**" type, originally intended for gate types only as a backward compatibility feature.

   This design choice makes of E-LOTOS a loosely typed language, where it is possible to declare a variable of type "**any**", which can be assigned any value. For instance, the following fragments of code are valid according to the CD:

   ```
   local var X:any in
       ?X := 143 ;
       ?X := "string" ;
       stop
   endloc
   ```

   and:

   ```
   local var X:any in
       if C then ?X := 143
       else ?X := "string"
   ```

```
        endif
    endloc
```

PROPOSED CHANGE:

As LOTOS, E-LOTOS should be strongly typed and type-checking should be done at compile-time only.

In particular, the use of the "**any**" type should be restricted to gate types only. The "**any**" type should not be available for ordinary variables.

ITEM: **AFNOR-7** **Simplifying patterns**

CLAUSE: General

QUALIFIER: Major Technical

RATIONALE:

The syntax of patterns proposed by the CD includes a so-called *pattern expression* feature "`!`*E*" (rule (P4) of Section 4.7, page 67).

This definition of patterns is very "original" (not to say "non-standard") compared to the way patterns are defined in other programming languages, including standard ML.

The main justification for this "original" definition of patterns is given page 26 : "*Th[e] use of patterns in communications is the main reason for allowing* `?` *and* `!` *in patterns*".

Page 18, an example of the use of "`!`" patterns is given (the palindrome example). However, this example is not convincing, because it could be written in shorter and simpler way:

```
if x = reverse (x) then "palindrome"
else "nonpalindrome"
endif
```

Although it attempts to unify classical patterns with experiment offers, this definition of patterns has several unpleasant effects:

- Syntactically, the "`?`" symbol is needed almost everywhere. For instance, each assignment must be written "`?x := 0`" instead of "`x := 0`".
- It is unclear whether the existing pattern-matching compiling algorithms (most of which assume that all variables in a pattern are free variables) can be adapted to patterns containing "`!`".
- The possibility of nesting patterns may lead to complex synchronization, in which "`?`" and "`!`" are combined at an arbitrary depth:

  ```
  G C1 (?X:int, C2 (!E, ?Y:int, C3 (!E)))
  ```

  The efficient implementation of such communication mechanism is an open problem. We are not aware of existing research in this direction.
- Finally, the "`!`" operators in patterns are redundant with the boolean guards that can be attached to the patterns in "**case**" expressions and actions. They are even less general than boolean guards, which allow to express non-functional relations between several variables.

PROPOSED CHANGE:

The attempt made by the CD at merging patterns and experiment offers is not the only way.

A much simpler solution exists, that is based on standard practice and that avoids the syntactic heaviness of the approach proposed by the CD:

- Keep the definition of patterns standard and simple : only constructors and free variables. Do not introduce "`!`" and "`?`" symbols.
- Keep experiment offers simple: an experiment offer can be either an expression E or a pattern preceded with a "`?`" symbol, e.g.:

```
G (1, "string", ?X:int, ?C(Y:int, Z:int))
```

ITEM: **AFNOR-8** **Unifying gates and signals**

CLAUSE: General

QUALIFIER: Major Technical

RATIONALE:

The language proposed in the CD introduces a new concept, the notion of *signal*, which does not exist in standard LOTOS.

There are several problems with the proposed approach.

Signals are very similar to *gates*. However, the CD lays out a number of syntactic and semantic differences between gates and signals:

- Gates and signals do not belong to the same identifier class.
- Signals can be used in expressions, whereas gates cannot.
- Signals can be caught using the "**trap**" operator, whereas gates cannot. Technically, this restriction is unnecessary, as the original proposal for the "**trap**" operator [GS96] was defined for gates.
- Gates can be synchronized by the parallel operator, whereas signals cannot.
- Gates are not urgent (they can let time pass).
- On the other hand, signals are always urgent, either if they are visible (i.e., declared at the top-level of the specification) or if they are declared in a "**trap**" operator. Notice that this feature goes against ET-LOTOS "philosophy" where a visible action can not be urgent
- In the CD, "**i**" is a signal (not a gate) and therefore urgent.
- However, the "δ" signal is urgent, except in parallel composition.

Technically, the distinction made between gates and signals is artificial. It are a major cause for the excessive complexity of E-LOTOS, especially in the dynamic semantics, where its is necessary to distinguish between many cases, depending if transitions are labelled with visible gates, "**i**" gate, signals, or "$i\delta$" signal (see the semantics of the "——" operator p. 83 for a striking example).

For the end-user, the existence of two concepts (gates and signals), which are similar to a large extent, but different and incompatible will be a major source of confusion.

When specifying the behaviour a real system, the user will have to decide whether each event should be represented as a gate or a signal. This is obvious from the Abracadabra example [QA92, GS96] where some events should be modeled as gates and other as signals. There

are no clear guidelines for making these design choices. Yet, these choices will affect the E-LOTOS description to be written, because of the restrictions laid on gates and signals.

Also, the existence of two concepts will go against the reusability of E-LOTOS process definitions. For instance, if a process P has been written using gates, it might be impossible to reuse P in the context of a "**trap**" operator. If it has been written using signals, it might be impossible to reuse it in the context of a parallel operator.

This is due to the fact that the restrictions concerning gates and signal go against compositionality of behaviours (which is really the ground of process algebras).

In this respect, it is worth mentioning that the Esterel language has only the notion of signal. Exception handling is done using a "**trap**" operator, which operates on signals. There is no reason why E-LOTOS should have a more complex approach.

PROPOSED CHANGE:

The revised CD should reduce the complexity of E-LOTOS as follows:

- Gates and signals should be unified in a a single concept (named signal in the sequel, but the terms actions or events are also possible).
- Value expressions remain unchanged: the evaluation of a value expression remains deterministic and can either yield a result value, or raise a signal.
- Behaviour expressions should be simplified by unifying gates and signals and by removing the "**signal**" operator, which is replaced by the action.
- As regards time and urgency, the following rules are sufficient:
  - Any visible signal is non-urgent
  - Any signal declared in a "**hide**" or "**trap**" operator becomes urgent

This approach will simplify E-LOTOS and make it closer to the ET-LOTOS language, a proposal whose time semantics has been proven to be consistent.

Notice that this proposal will allow parallel synchronization on the signals raised by expressions. For instance, if E_1 and E_2 are two expressions raising the signal A, the following behaviour expression is legal:

$$V_1 \texttt{ := } E_1 \texttt{ |[}A\texttt{]| } V2 \texttt{ := } E_2$$

This is not currently permitted by the CD; however, the same effect can be obtained by writing:

trap A->G **in** V_1 := E_1 **endtrap** |[A]| **trap** A->G **in** $V2$:= E_2 **endtrap**

Therefore, even under the restrictions defined by the CD, E-LOTOS implementors will be faced to synchronizations resulting from signals raised by expressions.

ITEM: **AFNOR-9** **Removing the "init" clause in the "local" construct**

CLAUSE: (B21) p. 50 and (E4) p. 51

QUALIFIER: Major Technical

RATIONALE:

The "**local**" constructs for behaviour expressions and expressions could be simplified by removing the "**init**" clause. We do not understand why this "**init**" was introduced in the CD :

- It does not exist in standard LOTOS
- It does not exist in imperative languages (C, Ada, Pascal, etc.) either
- It brings no expressivity, because :

  local var LV **init** E_1 **in** E_2 **endloc** $==$ **local var** LV **in** $(E_1 \ ; \ E_2)$ **endloc**

  and :

  local var LV **init** B_1 **in** B_2 **endloc** $==$ **local var** LV **in** $(B_1 \ ; \ B_2)$ **endloc**

PROPOSED CHANGE:

The "**init**" clause and the "**var**" keyword should be suppressed. The simplified syntax should be:

$$\textbf{local } LV \textbf{ in } E \textbf{ endloc}$$

and :

$$\textbf{local } LV \textbf{ in } B \textbf{ endloc}$$

In place of the "**init**" clause, the Committee may find suitable to keep the "**let**" construct that already exists in LOTOS. The "**let**" construct could be defined as a shorthand :

$$\textbf{let } X_1 : T_1 = E_1, ..., X_n : T_n = E_n \textbf{ in } E \textbf{ endlet} ==$$
$$\textbf{local } X_1 : T_1, ..., X_n : Tn \textbf{ in } X_1 := E_1; ...; X_n = E_n; E \textbf{ endloc}$$

and

$$\textbf{let } X_1 : T_1 = E_1, ..., X_n : T_n = E_n \textbf{ in } B \textbf{ endlet} ==$$
$$\textbf{local } X_1 : T_1, ..., X_n : Tn \textbf{ in } X_1 := E_1; ...; X_n = E_n; B \textbf{ endloc}$$

ITEM: **AFNOR-10** **Simplifying the "loop forever" construct**

CLAUSE: (B25) p. 50 and (E11) p. 51

QUALIFIER: Major Technical

RATIONALE:

The "**loop**" construct proposed by the CD can be improved in several ways:

- The "**forever**" keyword is useless from a syntactical point of view (it does not behave as a separator between two non-terminals). Moreover, this keyword is misleading if there is a "**break**" instruction to escape from the loop: in such case, the loop is not a "forever" loop.
- The "**init**" clause both non-standard (with respect to imperative languages) and useless, since we have :

  loop forever var LV **init** B_1 **in** B_2 **endloop** ==
  B_1; **loop forever var** LV **in** B_2 **endloop**

  and:

  loop forever var LV **init** E_1 **in** E_2 **endloop** ==
  E_1; **loop forever var** LV **in** E_2 **endloop**

- The "**var** LV" clause is confusing because the variables declared in LV are not the variables local to the loop, but the variables modified by the loop (i.e., modified in E_2 or B_2). This obligation of declaring the list of variables modified by a loop is not standard practice.
- It seems that static semantics of loops is unappropriate. For instance, the following expression type-checks statically but will provoke a run-time error:

  ```
  x := 1 ; loop forever var x:bool in x := x > 3
  ```

  The following expression type-checks, but the variable "`x`" used in "`x = x`" will change its type at run-time: during the first iteration, it is equal to `1`; during the second iteration, it is equal to `true`.

  ```
  x := 1 ; loop forever var x:bool in x := (x = x)
  ```

PROPOSED CHANGE:

The "**forever**", "**var**" and "**init**" clauses, which are useless or erroneous, should be removed. As in Ada, the "**loop**" construct should be put to its minimal form:

loop B **endloop**

ITEM: **AFNOR-11** **Improving breakable iterations**

CLAUSE: (B25) and (B31) p. 50, (E11) and (E12) p. 51, definition p. 99

QUALIFIER: Minor Technical

RATIONALE:

The CD introduces syntactic shorthand notations for breakable iterations: named loops, loops that return values, and a "**break**" instruction. These shorthand notations are defined in terms of "**loop forever**","**trap**" and "**raise**".

It is unclear whether these shorthand notations are worth being included in E-LOTOS: the shorthand notations have almost the same size as their expanded form (see p. 99).

Moreover, it is not sure that breaking nested loops (non-local exits) is a practice that needs to be encouraged by providing special constructs for doing so.

Finally, the proposed syntax is questionable. For instance, parentheses have to be doubled when writing "**break** $((1, 2))$".

PROPOSED CHANGE:

Before introducing the proposed shorthands for breakable iterations, other forms of loops should be considered.

In particular, the classical forms of loops found in Pascal, C, Ada ("**while** $X > 0$ ", "**for** $N :=$ 1 **to** 3", etc.) are probably more useful than the proposed breakable iteration.

ITEM: **AFNOR-12** **Removing the "exit" instructions from the user language**

CLAUSE: General

QUALIFIER: Major Technical

RATIONALE:

A well-known defect of LOTOS relies in its double form of sequential composition: on one hand the action prefix operator "**;**", on the other hand the "**exit**" and "**>>**" enable operators. These two forms of sequential composition are confusing to many users.

To solve this problem, it was proposed to equip E-LOTOS with a single form of sequential composition: the symmetrical operator "**;**", combined with features similar to those found in imperative languages (e.g., variable assignment and "**local**"constructs). There are many advantages in using the new E-LOTOS operators, which are both more powerful and closer to standard programming practice, which will favour the acceptance of E-LOTOS among a larger community.

However. it is clear that one should not keep simultaneously the old LOTOS operators while introducing the new E-LOTOS ones. This was agreed by the Committee during the Kansas City meeting: the minutes clearly express that: *the "***exit***" operator should disappear from* E-LOTOS.

Unfortunately, this is not the case in the CD: the new operators are introduced, but at the same time the old "**exit**" operator is kept.

The situation is even worse, because there are now 3 forms of "**exit**" statements:

- **exit**$[(RN)]$ (B3) p. 74
- **exit**(**any** T) (B4) p. 74
- **exit**(RE) (B29) p.75

It is interesting to notice that none of these three "**exit**" statements is as powerful as the existing "**exit**" of LOTOS, which allows to mix expressions and "**any**" values, e.g., "`exit (0, any bool)`".

The reason for such a mixture is historical:

- In the user language defined in the Kansas City output document, only the new E-LOTOS operators were available to the user.
- In the core language, "**exit**" was retained (and extended) as an auxiliary operator for defining the semantics of value-passing and continuations

- To prepare the CD, there has been a misguided attempt to merge the user language and the core language into a single language. Unfortunately, the approach followed was to make the union of the operators of the user and core languages.

PROPOSED CHANGE:

The simplest form of "**exit**" ("**exit**" without argument) should be kept in E-LOTOS, to play the role of the neutral element for sequential composition. It may be suitable to give it a better name, e.g., "**null**" as in Ada.

As the more complex forms of "**exit**" are only useful to define the semantics, they should not be directly available to the user.

Similarly, the form of "**process**" declarations with a "**exit**" clause (rule (D3) p. 58) can be removed as it is redundant with "**in/out**" parameters.

ITEM: **AFNOR-13** **Using a bracketed syntax**

CLAUSE: General

QUALIFIER: Minor Technical

RATIONALE:

The CD improves over standard LOTOS by adopting a bracketed syntax for several behaviour operators: "**local**", "**trap**", "**loop**"... All these operators are terminated with an "**endloc**", "**endtrap**", "**endloop**"... statement.

However, there remain many operators which are not properly bracketed. This is especially the case of infix binary operators "`;`", "`||`", "`[>`", etc.

This leads to ambiguity problems. For instance, how should the following expression be parsed:

$$B_1 \ ; \ B_2 \ || \ B_3$$

PROPOSED CHANGE:

There are two possible approaches to solve this problem:

- One could define operator precedences in order to solve parsing ambiguities. Of course, parentheses must be added in order to fight against fixed precedence. This is the solution used in standard LOTOS.

  However, this approach is not very satisfactory (should "`[]`" have precedence over "`[>`" or the opposite?) and is a frequent cause of mistakes in LOTOS.

- Another approach is to follow the bracketed syntax proposed by Ed Brinksma in his PhD thesis [Bri88]. The proposal here is an improvement of Ed Brinksma's proposal (the present solution is more user-friendly, and avoids bracketing as much as possible).

  We believe that it is possible to design a BNF syntax for E-LOTOS that would prevent the users from mixing different binary operators. In case of ambiguity, this syntax would force the user to use bracketing. The following guidelines should be taken as a goal:

  - As suggested by Brinksma, we introduce a special syntax to "bracket" all binary operators (except "`;`" which will be given the highest precedence):

    dis
    B_1
    `[>`
    B_2
    enddis

sel
B_0
[]
...
[]
B_n
endsel

etc.

Note: if the introduction of Brinkma's new keywords ("**sel**", "**dis**"...) is felt undesirable, they can be replaced with parentheses or "**begin**...**end**".

– As a general rule, mixing different binary operators is forbidden (it causes a syntax error). For instance, the following expression is illegal:

$$B_1 \ \texttt{[]} \ B_2 \ \texttt{[>} \ B_3$$

Instead, bracketing should be used. One should write either:

$$\textbf{sel} \ B_1 \ \texttt{[]} \ B_2 \ \textbf{endsel} \ \ \texttt{[>} B_3$$

or:

$$B_1 \ \texttt{[]} \ \textbf{dis} \ B_2 \ \texttt{[>} \ B_3 \ \textbf{enddis}$$

– However, there should be one exception to this principle: the ";" operator should have the highest precedence and need not be bracketed when combined with other operators. For instance, the following expression would be legal:

$$B_1 \ ; \ B_2 \ \texttt{[]} \ B_3$$

and parsed as:

$$(B_1 \ ; \ B_2) \ \texttt{[]} \ B_3$$

– Of course, when combining the same operator several times, bracketing should not be needed (assuming that the operator is left associative). For instance, the following expression is legal:

$$B_1 \ \texttt{[]} \ B_2 \ \texttt{[]} \ B_3$$

and equivalent to

$$\textbf{sel} \ B_1 \ \texttt{[]} \ B_2 \ \texttt{[]} \ B_3 \ \textbf{endsel}$$

ITEM: **AFNOR-14** **Improving the "write-once" variable scheme**

CLAUSE: General

QUALIFIER: Major Technical

RATIONALE:

The assignment operator proposed by the CD is a good step towards a user language that would be simpler and easier to use.

However, in its current form, this assignment operator can be misleading for users. The main reason for this is the following: the assignment operator combines a declaration of a new variable with an initialization.

For instance, when writing:

$$?x := x + 1; \textbf{ stop}$$

the x on the left-hand side is a new variable, not the same as the x on the right-hand side (this is the same situation as the "**let**" operator in LOTOS).

Similarly, it is possible to write expressions such as:

$$?x := (x > 1); \textbf{ stop}$$

where the left x has the boolean type and where the right x has the integer type.

PROPOSED CHANGE:

To address this problem, the revised CD should:

- evolve towards a write-many semantics
- clearly separate variable declaration from variable assignment
- ensure that the variables on the left and right hand side of an assignment are the same (thus, have the same type)
- ensure that the assigned variables have the same type as in their declaration

ITEM: **AFNOR-15** **Improving the "par" and "choice" operators**

CLAUSE: (B16) and (B19)

QUALIFIER: Major Technical

RATIONALE:

The "**choice**" and "**par**" operators proposed in the CD are not symmetrical:

- The "**choice**" operator can iterate on the domain of a type:

$$\textbf{choice}\ X : T\ \texttt{[]}\ B$$

- The "**par**" operator can only iterate on a list of values[1]:

$$\textbf{par}\ X\ \textbf{in}\ 1, 2, 3\ \texttt{|||}\ B$$

Moreover, only the full interleaving operator "`|||`" can be used with the proposed "**par**" operator: it may be useful to allow general parallel composition "`|[...]|`" as well as "n among m" synchronization.

PROPOSED CHANGE:

The revised CD should enforce a symmetry between "**choice**" and "**par**" operators:

1. The "**par**" operator should be able to iterate on finite types:

$$\textbf{par}\ X : T\ \texttt{|||}\ B$$

   Because types are defined constructively in E-LOTOS, it is very easy to determine statically whether a type is finite of not (this was not possible in LOTOS): a type is finite iff it is not recursive. This can be expressed with a simple fixed-point.

   Similarly, it is easy to compute inductively the list of values of a finite E-LOTOS type. The dynamic semantics could be simply defined by flattening the domain (as it is already done in LOTOS for the "**par**" operator over lists of gates).

   For the abstract data types, the definition of which is unknown, the problem is solved by adding a "**finite**" attribute to the definition of the abstract data type:

```
interface I is
   type T is finite
   ...
endtype
```

[1] At the time being, no rich-term syntax is provided for lists: one must use *cons* operators

This will allow generic modules to be checked statically. This approach will also be needed for generic processes, for which one has to indicate whether they are guarded or not.

2. If a "**par**" operator on a list of constants is found desirable, then a similar "**choice**" operator on a list of constants should be also introduced in E-LOTOS.

3. Obviously, the revised CD should not allow "**any**" in places such as "**choice** X : **any**...", which seem currently allowed by the CD. The need to restrict "**any**" types to gate types has already be explained in another AFNOR Major Technical comment.

ITEM: **AFNOR-16** **Improving the subtyping mechanism**

CLAUSE: General

QUALIFIER: Major Technical

RATIONALE:

The CD proposes a built-in subtyping mechanism for types. AFNOR makes the following comments:

1. It is clear that the E-LOTOS subtyping mechanism has no relation with the ODP subtyping mechanism.

   Recent work in this area [BBSDS97] establishes that ODP subtyping requires comparison relations between behaviour expressions corresponding to the service primitives provided by different ODP objects.

   In this respect, the notion of subtyping proposed by the CD (which is mainly the possibility to add fields into records) is totally unrelevant.

2. Technically, it is well-known that subtyping is incompatible with overloading. However, overloading is needed for the predefined type library and the rich-term syntax (a problem which has been left out of the CD). Therefore, when the Committee will define predefined types, subtyping will become a problem.

3. Anyway, the proposed subtyping mechanism is not very useful practically and raises a number of issues. The CD gives only two justifications for using built-in subtyping: subtyping between numeric types and subtyping for gate typing. Let us examine them in turn:

   - *Subtyping between numeric types:* the CD suggests that "**int**" could be a subtype of "**real**". It is probably better to avoid implicit conversions and let the user decide explicitly when integers should be converted to floating-point numbers.
   - *Subtyping between gate typing:* as far as gate typing is concerned, there are two ways of using the proposed subtyping:
     - Declaring a gate of type "**any**", meaning that the gate can accept any experiment offers
     - Declaring a gate of type *extensible record* "$(X_1 \Rightarrow T_1, ..., X_n \Rightarrow T_n, \mathbf{etc})$", meaning that the gate has at least n experiment offers of types $T_1, ..., T_n$, plus additional experiment offers of unspecified types.

   Let us observe that this mechanism is not as expressive as the original proposal for gate typing [Gar94a], which allows to specify a gate type with a fixed set of profiles:

```
channel T is
   (bool, int, int)
   (bool, bool)
endchan
```

This is not possible with the subtyping scheme proposed in the CD: the only way is to declare the gate of type "**any**", which will allow any combination of offers, e.g., `(int, bool)`.

4. The proposed subtyping is limited in the sense that it only supports *extensible records*. Practically, it is also desirable to have *extensible unions*. For instance, the following router accepts any kind of messages and sends them forward, except datagram packets, which are filtered:

```
type T is
    DATAGRAM (CODE:int)
  | etc
endtype

process ROUTER [INPUT:(T), OUTPUT:(T)] is
   loop
      INPUT ?X:T ;
      case X in
         DATAGRAM (CODE:int) -> exit
       | any T -> OUTPUT !X
      endcase
   endloop
endproc
```

(Below, we give hints about the way of allowing such a form of generalized subtyping in E-LOTOS).

5. Moreover, the proposed subtyping allows very questionable situations, e.g., recursive processes that extend their lists of parameters:

```
process P [G] (X => int, etc) : exit (none) is
   P [G] (X => 0)
   []
   P [G] (X => 0, Y => true)
   []
   P [G] (X => 0, Y => 1, Z => true)
   []
   P [G] (X => 0, Z => 1)
endproc
```

PROPOSED CHANGE:

The subtyping mechanism proposed by the CD is only arguable because major parts of E-LOTOS (predefined types and rich-term syntax) have been left out by the Committee. These parts remain to be provided. Because they require overloading, and because overloading and subtyping are incompatible, a conflict will occur.

- Instead of subtyping, gate typing and genericity should be used to achieve the desired effect, with the advantage that type-checking can be done statically. Let us consider the above example:

```
type T is
    DATAGRAM (CODE:int)
  | etc
endtype

process ROUTER [INPUT:(T), OUTPUT:(T)] is
   loop
      INPUT ?X:T ;
      case X in
         DATAGRAM (CODE:int) -> exit
       | any T -> OUTPUT !X
      endcase
   endloop
endproc
```

  In this definition, `T` is a formal type (in the same way as "**formalsorts**" in standard LOTOS) and `ROUTER` is a generic process parameterized with the formal type `T`.

  We suggest to follow the same approach as in LOTOS:

  - Generic definitions are checked statically: their syntax and static semantics are verified.
  - However, they are not given a dynamic semantics. As in LOTOS, only fully instantiated definitions are given a dynamic semantics. In the above example, process `ROUTER` has no dynamic semantics because it is parameterized.
  - Before being used, the `ROUTER` process must be instantiated, by substituting a concrete type (e.g., `PACKET`) to the formal type `T`. This is simply done using the existing module instantiation scheme.

    The instantiation of `ROUTER` will be a process in which `T` is replaced by `PACKET`. This instantiated process will have a dynamic semantics.

    During the instantiation, the static semantics must ensure that the `PACKET` type is a subtype of `T`, i.e., that `PACKET` has one constructor of the form `DATAGRAM (CODE:int)`.

- Also, in order to express ODP subtyping, AFNOR recommends to allow the expression of behavioural relations in E-LOTOS, exactly in the same way as equational specifications. Proposals already exist for this purpose.

References

[BBSDS97] H. Bowman, C. Briscoe-Smith, J. Derrick, and B. Strulo. On Behavioural Subtyping in LOTOS. In Howard Bowman and John Derrick, editors, *2nd IFIP International Conference on Formal Methods for Open Object-based Distributed Systems FMOODS'97 (Canterbury, UK)*, July 1997.

[Bri88] Ed Brinksma. *On the Design of Extended LOTOS, a Specification Language for Open Distributed Systems.* PhD thesis, University of Twente, November 1988.

[Gar94a] Hubert Garavel. On the Introduction of Gate Typing in E-LOTOS. Rapport SPECTRE 94-3, VERIMAG, Grenoble, February 1994. Annex D of ISO/IEC JTC1/SC21/WG1 N1314 Revised Draft on Enhancements to LOTOS and Annex B of ISO/IEC JTC1/SC21/WG1 N1349 Working Draft on Enhancements to LOTOS.

[Gar94b] Hubert Garavel. Six improvements to the process part of LOTOS. Rapport SPECTRE 94-7, VERIMAG, Grenoble, June 1994. Annex K of ISO/IEC JTC1/SC21/WG1 N1349 Working Draft on Enhancements to LOTOS.

[Gar95] Hubert Garavel. On the Introduction of Gate Typing in E-LOTOS. In Piotr Dembinski and Marek Sredniawa, editors, *Proceedings of the 15th IFIP International Workshop on Protocol Specification, Testing and Verification (Warsaw, Poland).* IFIP, Chapman & Hall, June 1995.

[GS96] Hubert Garavel and Mihaela Sighireanu. On the Introduction of Exceptions in LOTOS. In Reinhard Gotzhein and Jan Bredereke, editors, *Proceedings of the Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification FORTE/PSTV'96 (Kaiserslautern, Germany)*, pages 469–484. IFIP, Chapman & Hall, October 1996.

[Pec94] Charles Pecheur. A proposal for data types for E-LOTOS. Technical Report, University of Liège, October 1994. Annex H of ISO/IEC JTC1/SC21/WG1 N1349 Working Draft on Enhancements to LOTOS.

[Pec96] Charles Pecheur. *Improving the Specification of Data Types in* LOTOS. Doctorate thesis, University of Liège, November 1996.

[QA92] J. Quemada and A. Azcorra. Structuring Protocols with Exception in a LOTOS Extension. In *Proceedings of the 12th IFIP International Workshop on Protocol Specification, Testing and Verification (Orlando, Florida, USA).* IFIP, North-Holland, June 1992.