Characterizing Termination in LOTOS via Testing

David de Frutos[1], *Manuel Núñez*[1], *and Juan Quemada*[2]

[1]*Dept. Informática y Automática, Facultad de CC. Matemáticas*
Universidad Complutense de Madrid, E-28040 Madrid. Spain.
e-mail: {defrutos,manuelnu}@eucmvx.sim.ucm.es

[2]*Dept. Ingeniería Telemática, E.T.S.I. Telecomunicación*
Universidad Politécnica de Madrid, E-28040 Madrid. Spain.
e-mail: quemada@dit.upm.es

Abstract

We study termination of LOTOS behaviors. In order to characterize termination, we first define a testing semantics for LOTOS in the classical way giving a fully abstract characterization in terms of acceptance sets. We show that this testing semantics does not properly capture termination. Then we modify the testing semantics by extending the class of admissible tests, to obtain a new equivalence that is a congruence with respect to the enabling operator. Then, we study alternative definitions of testing in order to capture termination, and we relate them each other, getting a *Hierarchy* of termination, which besides proves that the first *termination* semantics is in fact the most adequate one.

Keywords

Distributed Systems; Process Algebras; LOTOS; Testing Semantics; Termination.

1 INTRODUCTION

Process Algebras (e.g. ACP (Bergstra, J. A. and Klop, J. W., 1984), CCS (Milner, R., 1980) and (Milner, R.,1989), CSP (Hoare, C.A.R., 1985)) have been widely used to specify concurrent systems. In this paper, we will work within the framework of another Process Algebra: LOTOS (Language of Temporal Ordering Specification) (ISO, 1988), which is a Formal Description Technique developed within ISO for the formal specification of open distributed system. Examples of its use can be found in (van Eijk, P., Diaz, M. and Vissers, C.A. editors, 1989).

Successful termination, expressed by means of an specific operator of the language can

* Research supported in part by the CICYT project TIC 94-0851-C02

be considered as one of the main new features of LOTOS. Successful termination also appears in CSP, see (Hoare, C.A.R., 1985), but looking at the posterior work on this process algebra, we see that not too much attention has been paid to this subject. This is mainly due to the greater attention paid to the prefix operator, compared to that paid to the sequential composition. In CCS we have no way to express at all successful termination, but this is justified by having no such a sequential composition operator. Finally in ACP successful termination is somehow expressed by default: There is no explicit operator to express it, but instead deadlock, that is unsuccessful termination is represented by a primitive process.

In the definition of LOTOS, successful termination is represented at the syntactical level by means of the *exit* behavior, while at the semantical level it is represented by event δ. Oppositely to what has happened for CSP, termination plays an important role on the application of LOTOS, mainly because of the wide use of the *enabling* operator.

There is not too much work devoted to the analysis of termination. Probably the most well known and thorough study is (Aceto, L. and Hennessy, M., 1992), where a denotational model capturing termination is derived from an operational semantics.

In this paper, we study how the termination of LOTOS programs may be characterized in the framework of testing semantics. Even if a testing semantics was defined for LOTOS in (Brinksma, E., Scollo, G. and Steenbergen, E., 1986), we have prefered to use some other definitions and notations, closer to those by (de Nicola, R. and Hennessy, M., 1984) and (Hennessy, M., 1988).

The rest of the paper is organized as follows. In Section 2 we review the syntax and operational semantics of LOTOS. In Section 3 we define a testing semantics for LOTOS. In Section 4 we modify the allowed test to capture termination of LOTOS behaviors. Finally, Section 5 presents our conclusions and future work.

2 SYNTAX AND OPERATIONAL SEMANTICS OF LOTOS

In this section we review the syntax and the operational semantics of LOTOS.

2.1 Syntax of LOTOS

As usual when giving the syntax of LOTOS, we will consider a universe of gates $\mathcal{G}$ that includes all the gate names; $g, g', g_1, \ldots$ range over $\mathcal{G}$. $G, G', G_1, \ldots$ range over sets of gate names. We will also consider an internal event i which is not visible for an observer and the termination event δ. Then the set $\mathcal{E} = \mathcal{G} \cup i \cup \delta$ will be the universe of events; $e, e', e_1, \ldots$ range over $\mathcal{E}$.

Definition 1 *$\mathcal{B}$ is the algebra of finite behavior expressions as defined in Table 1. Each behavior expression B has an associated gate set $L(B)$, also defined in the Table. $B, B', B_1, \ldots$ range over behavior expressions in $\mathcal{B}$.*

In Table 1, we have allowed as a valid behavior expression a process variable P, so we can define recursive definitions. We consider a finite index set I and the set of equations

Name	syntax	Gate set
Stop	stop	$\emptyset$
Exit	exit	$\{\delta\}$
Action Prefix	$e; B$	$L(B) \cup \{e\}$
Choice	$B_1 [] B_2$	$L(B_1) \cup L(B_2)$
Parallelism	$B_1 \mid\mid [G] \mid\mid B_2$	$L(B_1) \cup L(B_2)$
Enabling	$B_1 >> B_2$	$L(B_1) \cup L(B_2)$
Disabling	$B_1 [> B_2$	$L(B_1) \cup L(B_2)$
Process Variable	P	See comments below Def. 1
Hiding	hide G in B	$L(B) - G$
Relabeling	$B[g_1/g'_1, .., g_n/g'_n]$	$(L(B) - \{g'_1, .., g'_n\}) \cup \{g_1, .., g_n\}$

Table 1 Syntax of LOTOS.

$P_i := B_i$ $(i \in I)$, where P_i's are process variables and B_i's are behavior expressions that may contain process variables in the set $\{P_i \mid i \in I\}$. In this case the gate set of P_i is the gate set of B_i.

2.2 Operational Semantics

In this section, we present the rules which define the operational semantics of the language.

Exit, Prefix, Choice, and Process Variable

Termination

$$\textbf{[EXIT]} \quad \frac{}{\text{exit} \ -\delta\rightarrow \ \text{stop}}$$

Action Prefix

$$\textbf{[PRE]} \quad \frac{}{e; B - e \rightarrow B}$$

Choice

$$\textbf{[CHO1]} \quad \frac{B_1 - e \rightarrow B'_1}{B_1 [] B_2 - e \rightarrow B'_1} \qquad \textbf{[CHO2]} \quad \frac{B_2 - e \rightarrow B'_2}{B_1 [] B_2 - e \rightarrow B'_2}$$

Process Variable

$$\textbf{[REC]} \quad \frac{B - e \rightarrow B' \qquad P := B}{P - e \rightarrow B'}$$

Hiding and Relabeling

Hiding

$$\textbf{[HD1]} \quad \frac{B - e \rightarrow B' \qquad e \in G}{\text{hide } G \text{ in } B - i \rightarrow \text{hide } G \text{ in } B'}$$

$$\textbf{[HD2]} \quad \frac{B - e \rightarrow B' \qquad e \notin G}{\text{hide } G \text{ in } B - e \rightarrow \text{hide } G \text{ in } B'}$$

Relabeling

$$\textbf{[REL]} \qquad \frac{B - e \rightarrow B'}{B[g_1/g'_1, \ldots, g_n/g'_n] - e[g_1/g'_1, \ldots, g_n/g'_n] \rightarrow B'[g_1/g'_1, \ldots, g_n/g'_n]}$$

where $e[g_1/g'_1, \ldots, g_n/g'_n] = \begin{cases} g_i & \text{if } e = g'_i \\ e & \text{otherwise} \end{cases}$

Parallel Operator

The definition of the parallel operator includes two conceptually different types of operations: synchronization and interleaving.

Synchronization

$$\textbf{[SYN1]} \qquad \frac{B_1 - e \rightarrow B'_1 \qquad B_2 - e \rightarrow B'_2 \qquad e \in G \cup \delta}{B_1 \,|[G]|\, B_2 - e \rightarrow B'_1 \,|[G]|\, B'_2}$$

Interleaving

$$\textbf{[INT1]} \qquad \frac{B_1 - e \rightarrow B'_1 \qquad e \notin G \cup \delta}{B_1 \,|[G]|\, B_2 - e \rightarrow B'_1 \,|[G]|\, B_2}$$

$$\textbf{[INT2]} \qquad \frac{B_2 - e \rightarrow B'_2 \qquad e \notin G \cup \delta}{B_1 \,|[G]|\, B_2 - e \rightarrow B_1 \,|[G]|\, B'_2}$$

Disabling and Enabling

Disabling

$$\textbf{[DIS1]} \qquad \frac{B_1 - e \rightarrow B'_1 \qquad e \neq \delta}{B_1 \,[>\, B_2 - e \rightarrow B'_1 \,[>\, B_2} \qquad \textbf{[DIS2]} \qquad \frac{B_1 - \delta \rightarrow B'_1}{B_1 \,[>\, B_2 - \delta \rightarrow B'_1}$$

$$\textbf{[DIS3]} \qquad \frac{B_2 - e \rightarrow B'_2}{B_1 \,[>\, B_2 - e \rightarrow B'_2}$$

Enabling

$$\textbf{[ENA1]} \qquad \frac{B_1 - e \rightarrow B'_1 \qquad e \neq \delta}{B_1 >> B_2 - e \rightarrow B'_1 >> B_2}$$

$$\textbf{[ENA2]} \qquad \frac{B_1 - \delta \rightarrow B'_1}{B_1 >> B_2 - i \rightarrow B_2}$$

3 A *PLAIN* TESTING SEMANTICS FOR LOTOS

Once we have defined the operational semantics, we can define the testing semantics. In order to give a formal definition of testing, we need first to introduce some new concepts and definitions. To indicate successful test termination, tests will use a new event *ok*, which can only appear in tests. Then we can define the notion of complete computations.

Definition 2 *:* (Complete computations) *We say that a (possibly infinite) sequence of transitions* $B - e_1 \rightarrow B_1, B_1 - e_2 \rightarrow B_2, \ldots, B_{n-1} - e_n \rightarrow B_n$ *is a complete computation of* B *, when if fulfills any of the followings conditions:*

1. *It is successful which means that it is finite,* $e_n = ok$ *and* $e_i \neq ok \ \forall\, i < n$.
2. *It is blocked which means that* B_n *can generate no more transitions.*
3. *It is infinite and unsuccessful, which means that* $e_i \neq ok \ \forall\, i \in I\!N$.

In this section tests are just LOTOS behaviors, but possibly including the new acceptance event *ok* . Testing is performed by studying the computations of the behavior expression that is obtained by putting a test in parallel with the behavior expression to be tested, taking as synchronizing set the full alphabet but the acceptance event *ok* .

Test application will be represented by $B|T$, and is defined as follows:

$$B|T = hide\ \mathcal{G}\ in\ (B|[\mathcal{G}]|T)$$

Now, satisfaction of tests can be defined in a very similar way to that used in (de Nicola, R. and Hennessy, M., 1984) and (Hennessy, M., 1988).

Definition 3 *We say that* B *may satisfy a test* T*, and we will write* $B\ may\ T$*, iff* $B|T$ *has some successful computation. We say that* B *must satisfy a test* T*, and we will write* $B\ must\ T$*, iff any complete computation of* $B|T$ *is a successful computation.*

Definition 4 *If* B_1 *and* B_2 *are behavior expressions, we say that they are:*

- *may-equivalent* $B_1 \simeq_{\text{may}} B_2$*, if they satisfy the same may tests*

$$\forall T \quad B_1\ may\ T \ \Leftrightarrow B_2\ may\ T$$

- *must-equivalent* $B_1 \simeq_{\text{must}} B_2$*, if they satisfy the same must tests*

$$\forall T \quad B_1\ must\ T \ \Leftrightarrow B_2\ must\ T$$

- *maymust-equivalent* $B_1 \simeq B_2$*, if they are both may and must-equivalent.*

Taking as tests plain LOTOS behaviors, just extended with the acceptance event *ok* , termination can only be partially captured. For instance, we have

$$\text{exit} \ \not\simeq_{\text{must}} \ \text{stop}$$

since taking $T = i; ok \ [] \ \text{exit}$ we have stop must T but exit $\not$must T. However

$$\text{exit} \ [] \ i; \text{stop} \ \simeq_{\text{must}} \ \text{exit}$$

This is due to the fact that tests cannot succeed after the termination of the testing behavior. Thus we can test the possibility of termination, but not the obligation to do it.

A technical problem appears as a consequence of the fact above: the enabling operator does not preserve must-equivalence, since, for instance, we have

$$(\text{ exit } [] \; i;\ \text{stop }) >> (g;\ \text{stop }) \not\simeq_{\text{must}} \text{ exit } >> (g;\ \text{stop })$$

for any gate name g. Thus we need to make some changes in the definition of the testing semantics, if we want to capture termination in the adequate way. But first we will present an alternative characterization of the testing semantics, which will be useful in order to define the desired new definition of testing.

3.1 An alternative characterization

According to the definition of testing semantics, to check the equivalence between behavior expressions, one should have to see that they pass exactly the same tests; but the number of tests is infinite. That is why we need an alternative characterization of the testing semantics. This characterization is obtained following (de Nicola, R. and Hennessy, M., 1984) and (Hennessy, M., 1988), and it is based on the operational semantics of the language, and thus it is independent of its syntax.

In the rest of the paper we restrict ourselves to the study of the $\simeq_{\text{must}}$ equivalence. First, we introduce some auxiliary definitions.

Definition 5 *We define the set of visible events $\mathcal{V}$ as the set $\mathcal{G} \cup \delta$; $v, v', v_1 \ldots$ range over $\mathcal{V}$. $V, V', V_1, \ldots$ range over sets of visible events.*

Definition 6 *Let B, B' behavior expressions.*

We (inductively) define $B \rightarrow^* B'$ as:

1. $B \rightarrow^* B$
2. If $B - i \rightarrow B_1 \wedge B_1 \rightarrow^* B'$ then $B \rightarrow^* B'$

Intuitively speaking, $B \rightarrow^* B'$ means that B evolves to B' by executing a sequence (possibly empty) of internal events.

We (inductively) define $B = s \Rightarrow B'$ as:

1. $B = \epsilon \Rightarrow B'$ iff $B \rightarrow^* B'$
2. $B = v\, s' \Rightarrow B'$ iff $\exists B_1, B_2 : B \rightarrow^* B_1 - v \rightarrow B_2 \wedge B_2 = s' \Rightarrow B'$

where s, s' range over sequences of visible events, and ϵ represents the empty sequence. Intuitively speaking, $B = s \Rightarrow B'$ if B evolves to B' by executing a sequence of internal events, then the first visible event of the sequence s, then a sequence of internal events and so on. We write $B = s \Rightarrow$ if there exists B' such that $B = s \Rightarrow B'$.

Definition 7 *Let s, s' be sequences of visible events. We say that $s' \leq s$ if s' is a prefix of s. If $s' \leq s$, we define $s \stackrel{\bullet}{-} s'$ as the sequence t, s.t. $s = s'\hat{}t$.*

Definition 8 *(Acceptance Sets (Hennessy, M., 1988))*
For any behavior expression B, and any sequence of visible events s, we define

- $S(B) = \{v \mid B = v \Rightarrow\}$.
- $\mathcal{A}(B, s) = \{S(B') \mid B = s \Rightarrow B'\}$.

As in (Hennessy, M., 1988), we write $\mathcal{A}(B_1, s) \ll \mathcal{A}(B_2, s)$ iff for every $A \in \mathcal{A}(B_2, s)$ there is some $A' \in \mathcal{A}(B_1, s)$ such that $A' \subseteq A$.

For nonterminating LOTOS behaviors (that are those that cannot execute the termination event δ), we can present the announced alternative characterization of testing semantics.

Definition 9 *For behavior expressions B_1 and B_2, we write $B_1 \approx B_2$ iff for every $s \in \mathcal{V}^*$ we have*

$$\mathcal{A}(B_1, s) \ll \mathcal{A}(B_2, s) \text{ and } \mathcal{A}(B_2, s) \ll \mathcal{A}(B_1, s)$$

Theorem 1 *Let B_1, B_2 be nonterminating LOTOS behaviors. Then we have $B_1 \approx B_2$ iff $B_1 \simeq_{\text{must}} B_2$.*

Proof. Similar to that in (Hennessy, M., 1988). □

This characterization is not valid anymore, if we consider behaviors which may terminate. For instance, for the behaviors which we considered just before the beginning of this subsection, we have

$$\text{exit } [] \ i;\ \text{stop } \not\approx \text{ exit}$$

In fact, we will see that the new concept of testing which we introduce in the next section is indeed characterized by the previous alternative definition.

4 A TESTING SEMANTICS CAPTURING TERMINATION

Our aim is to define tests with the ability of controlling successful termination of LOTOS behaviors. We have several possibilities; the first one is to consider that event δ can appear in tests, and besides it is treated by the parallel operator connecting the tested behavior and the corresponding test, as any ordinary event.

Besides we want that the testing equivalence will be a congruence with respect to enabling, that is to say $B_1 \sim B_2 \Rightarrow (B_1 >> B_3) \sim (B_2 >> B_3)$. We will see that it is guaranteed if we allow event δ to appear in tests. We will call $\delta-$test to those containing event δ, and we denote this set of tests by $Test_\delta$. When we use this class of tests, we have a new equivalence relation between behaviors.

Definition 10 *Let B_1, B_2 be behavior expressions. We say that they are must$_\delta$-equivalent ($B_1 \simeq_{\text{must}_\delta} B_2$), if they must pass the same $\delta-$tests*

$$\forall\, T \in Test_\delta \quad B_1 \; must \; T \;\Leftrightarrow B_2 \; must \; T$$

Obviously, for any behaviors B, B' we have $B \simeq_{\text{must}_\delta} B' \Rightarrow B \simeq_{\text{must}} B'$. We also have the result that follows

Corollary 1 *Let B_1, B_2 nonterminating behaviors. Then*

$$B_1 \simeq_{\text{must}} B_2 \Leftrightarrow B_1 \simeq_{\text{must}_\delta} B_2$$

With this definition, we have exit [] i; stop $\not\simeq_{\text{must}_\delta}$ exit because test $T = \delta; ok$ distinguishes them. Now we can extend Theorem 1 to any pair of LOTOS behaviors.

Theorem 2 *Let B_1, B_2 be LOTOS behaviors. Then $B_1 \approx B_2$ iff $B_1 \simeq_{\text{must}_\delta} B_2$.*

Proof. Analogous to that of Theorem 1, because once tests can both accept of reject after the execution of the termination event δ, this event is treated in the same way that any ordinary event. □

In the following we prove that this must equivalence is in fact a congruence with respect to the enabling operator. First we need a lemma concerning acceptance sets for the case of the enabling operator $\mathcal{A}(B{>>}B', s)$.

Lemma 1 *Given B_1, B_2 behavior expressions and s a sequence of visible events, we have:*

$$\begin{aligned}\mathcal{A}(B_1{>>}B_2, s) \;&= \{S(P) \mid s' \leq s \;\wedge\; B_1 = s'\delta \Rightarrow \;\wedge\; B_2 = s \stackrel{\bullet}{-} s' \Rightarrow P\} \\ &\cup \{S(P) \mid B_1 = s \Rightarrow P \wedge\; \delta \notin s\} \qquad (1)\end{aligned}$$

Proof. s may be a trace of $B_1{>>}B_2$ or not. First let us suppose that the former holds.

Then, the only possible computations from $B_1{>>}B_2$ such that $B_1{>>}B_2 = s \Rightarrow P$ are either like $B_1{>>}B_2 = s' \Rightarrow P'{>>}B_2 {-} i \rightarrow B_2 {=} s \stackrel{\bullet}{-} s' \Rightarrow P$ or $B_1{>>}B_2 \;= s \;\Rightarrow P'{>>}B_2$. And since $\mathcal{A}(B_1{>>}B_2, s) = \{S(P) \mid B_1{>>}\; B_2 \;=\; s \;\Rightarrow\; P\}$, then (1) easily follows, taking into account the remark above defining the possible shape of the computations of $B_1{>>}B_2$.

Now let us suppose that s is not a trace of $B_1{>>}B_2$. Then, $\mathcal{A}(B_1{>>}B_2, s) = \emptyset$.

Besides, if s is not a trace of $B_1{>>}B_2$, then it is neither a trace of B_1, and it follows that $\{S(P) \mid B_1 = s \Rightarrow P \;\wedge\; \delta \notin s\} = \emptyset$. Also, if s is not a trace of $B_1{>>}B_2$ there are no t, t' such that $s = t\hat{}t'$, $B_1 = t\delta \Rightarrow$ and $B_2 = t' \Rightarrow$. Then we infer that the sets in $\{S(P) \mid B_1 = s'\delta \Rightarrow \;\wedge\; B_2 = s \stackrel{\bullet}{-} s_i \Rightarrow P\}$ are empty. From these two facts we conclude that the set at the right hand side of Equation (1) is the empty set. □

Lemma 2 *Let B_1, B_2 be behavior expressions. If $B_1 \simeq_{\text{must}_\delta} B_2$ then for any trace s*

$$(B_1 = s\delta \Rightarrow) \;\Leftrightarrow\; (B_2 = s\delta \Rightarrow)$$

Theorem 3 *If $B_1 \simeq_{\mathrm{must}_\delta} B_2$ then for any behavior expression B, the following two conditions hold:*

- $B_1 >> B \simeq_{\mathrm{must}_\delta} B_2 >> B$.
- $B >> B_1 \simeq_{\mathrm{must}_\delta} B >> B_2$.

Proof. We will only prove the first condition, since the second one is very similar.

Due to Theorem 2 we can use $\approx$ instead of $\simeq_{\mathrm{must}_\delta}$. Then we have to prove that for any s, $\mathcal{A}(B_2>>B, s) \ll \mathcal{A}(B_1>>B, s)$ and $\mathcal{A}(B_1>>B, s) \ll \mathcal{A}(B_2>>B, s)$. Obviously it is enough to prove one of them, since they are symmetric.

Let $A \in \mathcal{A}(B_1>>B, s)$ we have to show that there exists some $C \in \mathcal{A}(B_2>>B, s)$ such that $C \subseteq A$. By Lemma 1 $A \in \{S(P) \mid s' \leq s \ \wedge \ B_1 = s'\delta \Rightarrow \ \wedge \ B = s \stackrel{\bullet}{-} s' \Rightarrow P\}$ or $A \in \{S(P) \mid B_1 = s \Rightarrow P \wedge \ \delta \notin s\}$.

Suppose $A \in \{S(P) \mid s' \leq s \ \wedge \ B_1 = s'\delta \Rightarrow \ \wedge \ B = s \stackrel{\bullet}{-} s' \Rightarrow P\}$. By Lemma 2 we have

$$\{S(P) \mid s' \leq s \ \wedge \ B_1 = s'\delta \Rightarrow \ \wedge \ B = s \stackrel{\bullet}{-} s' \Rightarrow P\} =$$
$$\{S(P) \mid s' \leq s \ \wedge \ B_2 = s'\delta \Rightarrow \ \wedge \ B = s \stackrel{\bullet}{-} s' \Rightarrow P\}$$

and then $A \in \{S(P) \mid s' \leq s \ \wedge \ B_2 = s'\delta \Rightarrow \ \wedge \ B = s \stackrel{\bullet}{-} s' \Rightarrow P\} \subseteq \mathcal{A}(B_2>>B, s)$.

Now suppose $A \in \{S(P) \mid B_1 = s \Rightarrow P \ \wedge \ \delta \notin s\} \subseteq \mathcal{A}(B_1, s)$. If we now apply that $\mathcal{A}(B_2, s) \ll \mathcal{A}(B_1, s)$ we conclude that there exists some $C \in \mathcal{A}(B_2, s)$ such that $C \subseteq A$, and $C = S(P')$ for a behavior P' s.t. $B_2 = s \Rightarrow P'$. Obviously, $C \in \mathcal{A}(B_2>>B, s)$. □

In fact, $\simeq_{\mathrm{must}_\delta}$ is the greatest congruence contained in $\simeq_{\mathrm{must}}$. To prove it, we have to introduce a previous definition.

Definition 11 *Let tic be a new event which cannot appear in behaviors (but it can appear in tests). Let B_1, B_2 be behaviors. We define the equivalence relation $\simeq_{\mathrm{must}_{tic}}$*

$$B_1 \simeq_{\mathrm{must}_{tic}} B_2 \ \Leftrightarrow \ B_1 >> Tic \ \simeq_{\mathrm{must}_\delta} B_2 >> Tic$$

where $Tic = tic$; stop .

Remark: Since the processes in both sides of the definition ($B_1>>Tic$ and $B_2>>Tic$) do not terminate, the definition does not change if we substitute in it $\simeq_{\mathrm{must}_\delta}$ by any equivalence relation which coincides with $\simeq_{\mathrm{must}_\delta}$ on nonterminating processes (for example $\simeq_{\mathrm{must}}$).

Theorem 4 $\simeq_{\mathrm{must}_{tic}} \cap \simeq_{\mathrm{must}} = \simeq_{\mathrm{must}_\delta}$

Proof. The right to left inclusion is a consequence of Theorem 3.

For the left to right inclusion, we use the characterization of $\simeq_{\mathrm{must}_\delta}$ by acceptance sets (see Theorem 2). Acceptance sets of a behavior $B>>Tic$ can be obtained from the acceptance sets of B in the following way:

$$\begin{array}{lcll} \mathcal{A}(B{>>}Tic, s) & = & \mathcal{A}(B, s)^{tic}_{\delta} & \text{(if } s \text{ does not contain neither } \delta \text{ nor } tic\text{)} \\ \mathcal{A}(B{>>}Tic, s\,tic) & = & \mathcal{A}(B, s\,\delta) & \end{array}$$

for any trace s, where S^{tic}_{δ} is equal to the state S but replacing δ (if $\delta \in S$) by tic.

We will reason by contradiction. Suppose that there exist behavior expressions B_1, B_2 such that $B_1 \simeq_{\text{must}} B_2 \wedge B_1 \simeq_{\text{must}_{tic}} B_2$. Then, since the characterization by acceptance sets is also valid for $\simeq_{\text{must}}$ when restricted to nonterminating behaviors (see Theorem 1), we have

$$\mathcal{A}(B_1{>>}Tic, s) \ll \mathcal{A}(B_2{>>}Tic, s) \ \wedge \ \mathcal{A}(B_2{>>}Tic, s) \ll \mathcal{A}(B_1{>>}Tic, s)$$

for any trace s.

Now suppose that $B_1 \not\simeq_{\text{must}_{\delta}} B_2$. Then, using the alternative characterization given in Theorem 2, we have:

$$\exists s' : \mathcal{A}(B_2, s') \not\ll \mathcal{A}(B_1, s') \ \vee \ \mathcal{A}(B_1, s') \not\ll \mathcal{A}(B_2, s')$$

Without lose of generality, suppose that the former holds. Then, $\exists V' \in \mathcal{A}(B_1, s')$ such that $\forall V_i \in \mathcal{A}(B_2, s')$, we have $V_i \not\subseteq V'$. Note that δ must be in V', because $B_1 \simeq_{\text{must}} B_2$ and the characterization by acceptance trees is valid for $\simeq_{\text{must}}$ when δ does not appear in acceptance trees.

Then, $\forall V_i \in \mathcal{A}(B_2, s') : \exists v_i \in V_i - V'$. Note that $v_i \neq \delta$ (see previous paragraph). Now we consider the test $T = \tilde{s}'; (v_1; ok \ [] v_2; ok \ [] \cdots v_K; ok)$, where if $s' = \langle g_1 g_2 \ldots g_r \rangle$, then $\tilde{s}' = g_1; g_2; \ldots g_r$. Note that in the test T does not appear δ. Finally we have $B_2 \ must\ T$, but $B_1 \not\!must T$, and then $B_1 \not\simeq_{\text{must}} B_2$ which contradicts our hypothesis. □

Corollary 2 $\simeq_{\text{must}_{\delta}}$ *is the greatest congruence with respect to the enabling operator, contained in* $\simeq_{\text{must}}$.

Proof. Taking into account the definition of $\simeq_{\text{must}_{tic}}$, we have that any equivalence relation preserved by enabling must be contained in $\simeq_{\text{must}_{tic}}$. □

4.1 Alternative Definitions

There are other possibilities to define testing of successful termination. For instance we will discuss what happen if we do not allow event δ to appear in tests, but instead we allow the behavior *exit* to appear in them. This implies that δ can still be performed, but when that happen no further step can be done. Then, we can consider that when δ is executed, the test has failed or we can consider that the test has succeeded. We also consider when the *exit* behavior cannot appear in tests.

δ means Failure

Since in this case the acceptance event cannot be executed after the termination event δ, we have that whenever this last event can be performed by the pair (behavior expression, test), then the test cannot be passed. With this interpretation we get just the equivalence $\simeq_{\text{must}}$.

δ means ok

A second possibility is to take any (successfully) terminating computation as successful. For this, we have just to change the definition of successful computation, introducing as new successful computations those executing δ.

This new way of defining the set of successful computations generates a new must-semantics, whose defining equivalence relation we will denote by $\simeq_{\text{must}_1}$. Now we have a result similar to that of Corollary 1.

Corollary 3 *If B_1, B_2 are two behavior expressions that cannot execute δ, then*

$$B_1 \simeq_{\text{must}_1} B_2 \;\Leftrightarrow\; B_1 \simeq_{\text{must}_\delta} B_2 \;\; (\Leftrightarrow \;\; B_1 \simeq_{\text{must}} B_2)$$

Proof. Obviously the appearance of δ in tests is useless when comparing two behaviors which cannot terminate. □

Like in the case of $\simeq_{\text{must}}$, this new equivalence is not preserved by enabling, as the following example shows.

Example 1 *The behavior expressions B_1 and B_2 defined by $B_1 = i$; stop [] i; exit and B_2 =stop , are must$_1$-equivalent, but if we consider the behavior expressions B_1>>(g; exit) and B_2>>(g; exit) for any gate name g, then we have that they are not equivalent, because test $T = i; ok$ [] g; stop distinguishes them (the former behavior $\not{must}_1$ this test, while the latter must$_1$ it).*

Since $\simeq_{\text{must}_1}$ coincides with $\simeq_{\text{must}}$ when restricted to nonterminating behaviors, we immediately have similar results to that in Theorem 4 and Corollary 2.

Theorem 5 $\simeq_{\text{must}_{tic}} \cap \simeq_{\text{must}_1} = \simeq_{\text{must}_\delta}$

Corollary 4 $\simeq_{\text{must}_\delta}$ *is the greatest congruence with respect to the enabling operator, contained in* $\simeq_{\text{must}_1}$.

Let us remark that stop $\not\simeq_{\text{must}_1}$ exit because the test *exit* distinguishes them. The following lemma shows that there are no relation between $\simeq_{\text{must}}$ and $\simeq_{\text{must}_1}$.

Lemma 3 *Neither* $\simeq_{\text{must}} \subseteq \simeq_{\text{must}_1}$ *nor* $\simeq_{\text{must}_1} \subseteq \simeq_{\text{must}}$.

Proof. $\simeq_{\text{must}} \not\subseteq \simeq_{\text{must}_1}$ because exit $\simeq_{\text{must}}$ i; stop [] i; exit but they are not must$_1$-equivalent because of the test $T =$ exit (the former behavior *must*$_1$ it while the latter $\not{must}_1$ it).

$\simeq_{\text{must}_1} \not\subseteq \simeq_{\text{must}}$ because stop $\simeq_{\text{must}_1}$ i; stop [] i; exit but they are not must-equivalent because of the test $T =$ exit [] $i; ok$ (the former behavior *must* it while the latter $\not{must}$ it). □

Even though there is no relation between $\simeq_{\text{must}}$ and $\simeq_{\text{must}_1}$ (see Lemma 3), we have an interesting result which relates the combination of these orders with $\simeq_{\text{must}_\delta}$.

Theorem 6 $B_1 \simeq_{\text{must}} B_2 \wedge B_1 \simeq_{\text{must}_1} B_2 \Leftrightarrow B_1 \simeq_{\text{must}_\delta} B_2$

Proof. (Sketch) The family of basic tests which is used in (Hennessy, M., 1988) to characterize the must-semantics, contains two classes of tests. In the first class (trace tests) we have to refuse after the execution of a trace; while in the other one (refusal tests) we accept after any of the gate names in an offered set. But tests containing both refusals and acceptances after the execution of an observable event are not necessary to characterize the must-semantics. In our case, *trace tests* can be obtained when we consider that δ *means* failure (thus, this set of tests characterizes the must-equivalence), while *refusal tests* can be obtained when we consider that δ *means ok* (thus, this set of tests characterizes the must_1-equivalence). □

Neither δ nor exit in tests

Again we consider that δ is a special event but now it cannot be generated by tests (that implies that *exit* cannot appear in tests). We will denote the corresponding set of tests by $Test_{-exit}$. Then, as in the previous cases, we can define a new equivalence relation.

Definition 12 *Let B_1, B_2 be behavior expressions. We say that they are $must_2$-equivalent ($B_1 \simeq_{\text{must}_2} B_2$), if they must pass the same tests in $Test_{-exit}$*

$$\forall T \in Test_{-exit} \quad B_1 \; must \; T \;\Leftrightarrow\; B_2 \; must \; T$$

But again this equivalence is not a congruence as the next example shows

Example 2 *The behavior expression B_1, B_2 defined by $B_1 = exit$ and $B_2 = stop$ are $must_2$ equivalent. But if we consider the behavior expressions $B_1 >> (g;\, exit)$ and $B_2 >> (g;\, exit)$ for any gate name g, we have that they are not equivalent, because the test $g; ok$ distinguishes them (the former must this test, but the latter must it).*

For the same reason that in the previous case, we have for this equivalence relation the same results that in Theorem 4 and Corollary 2.

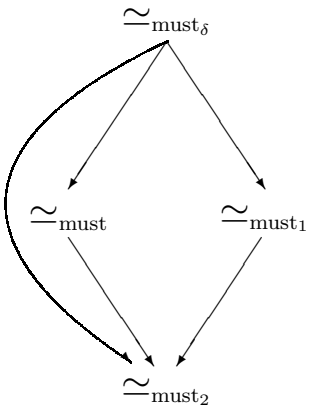
Theorem 7 $\simeq_{\text{must}_{tic}} \cap \simeq_{\text{must}_2} = \simeq_{\text{must}_\delta}$

Corollary 5 $\simeq_{\text{must}_\delta}$ *is the greatest congruence with respect to the enabling operator, contained in* $\simeq_{\text{must}_2}$.

Let us remark that this relation is different from $\simeq_{\text{must}}$ and $\simeq_{\text{must}_1}$, since, for instance, exit $\simeq_{\text{must}_2}$ stop , but exit $\not\simeq_{\text{must}}$ stop , and exit $\not\simeq_{\text{must}_1}$ stop . In fact, $\simeq_{\text{must}_2}$ is *weaker* than the equivalences previously presented.

The following theorem relates the equivalences presented in this paper.

Theorem 8 (Hierarchy of Termination) *Let B_1, B_2 be behavior expressions.*



- $B_1 \simeq_{\text{must}_\delta} B_2 \Rightarrow B_1 \simeq_{\text{must}} B_2$
- $B_1 \simeq_{\text{must}_\delta} B_2 \Rightarrow B_1 \simeq_{\text{must}_1} B_2$
- $B_1 \simeq_{\text{must}_\delta} B_2 \Rightarrow B_1 \simeq_{\text{must}_2} B_2$
- $B_1 \simeq_{\text{must}} B_2 \Rightarrow B_1 \simeq_{\text{must}_2} B_2$
- $B_1 \simeq_{\text{must}_1} B_2 \Rightarrow B_1 \simeq_{\text{must}_2} B_2$

5 CONCLUSION AND FUTURE WORK

We have studied termination of LOTOS behaviors in the framework of a testing semantics. First, we have defined the testing semantics in the classical way (Hennessy, M., 1988). But with this definition, we cannot properly capture termination of behaviors; more exactly, we can detect the possibility of termination, but not the obligation to do it. Following (Hennessy, M., 1988) we present a characterization based on acceptance trees, of the given semantics, which is equivalent to the must-equivalence when restricted to nonterminating LOTOS behaviors.

In order to capture termination, we modify the testing semantics by extending the class of admissible tests, allowing action δ to appear in tests. We get a new equivalence relation $\simeq_{\text{must}_\delta}$, which is proved to be a congruence with respect to the enabling operator, and such that it is fully characterized by acceptance trees.

Since it is not totally clear that the proposed way to test termination will be the *natural* one, we have studied some alternative definitions of testing in order to capture termination. All of them are extensions of the standard way to test nonterminating behaviors. Thus the differences are concentrated on the way to treat termination. With the standard definition of testing, termination can only be tested by means of the terminating behavior *exit*, and thus termination means failure, since it cannot be followed by the acceptance event *ok*. The induced equivalence is not a congruence with respect to enabling, and in fact, we have that the greatest congruence contained in that relation is just $\simeq_{\text{must}_\delta}$.

Exactly the same results are obtained when we change the definition, either taking that termination means success, or when we do not allow to test termination at all. In our opinion, from all these results we can conclude that to test the termination event as any other ordinary event is indeed the appropriate way to test termination.

As future work, we plan to extend this study to the case in which there is not a single termination event, but a set of different termination events, as proposed in (Quemada, J., 1994). We also would like to study termination in the framework of probabilistic and timed extensions of LOTOS.

REFERENCES

Aceto, L. and Hennessy, M. (1992) Termination, deadlock, and divergence. *Journal of the Association for Computer Machinery*, **39(1)**:147–187.

Bergstra, J. A. and Klop, J. W. (1984) Process algebra for syncronous communication. *Information and Control*, **60**:109–137.

Brinksma, E., Scollo, G. and Steenbergen, E. (1986) LOTOS specifications, their implementations and their tests. In Proceedings of the Sixth Symposium on *Protocol Specification, Testing and Verification*, 349–360.

de Nicola, R. and Hennessy, M. (1984) Testing equivalences for processes. *Theoretical Computer Science*, **34**:83–133.

Hennessy, M. (1988) *Algebraic Theory of Processes*. MIT Press.

Hoare, C.A.R. (1985) *Communicating Sequential Processes*. Prentice Hall.

ISO (1988) LOTOS. A formal description technique based on the temporal ordering of observational behaviour. IS 8807, TC97/SC21.

Milner, R. (1980) *A Calculus of Communicating Systems*. Lecture Notes in Computer Science **92**. Springer-Verlag.

Milner, R. (1989) *Communication and Concurrency*. Prentice Hall.

Quemada, J. (1994) Generalized termination and enabling. In *Working Draft on Enhancements to LOTOS (Annex B)*. ISO/IEC JTC1/SC21/WG1.

van Eijk, P., Diaz, M. and Vissers, C.A. editors (1989). *The Formal Description Technique LOTOS*. North-Holland.

Prof. David de Frutos graduated in “Pure Mathematics and Computer Science” in 1981 from the Universidad Complutense de Madrid (UCM) and achieved a PhD in ”Mathematics (Computer Science)” in 1985 with a Thesis devoted to “Denotational Semantics of Probabilistic Constructions (Probabilistic Powerdomains)”. He was Associate Professor of Computer Science from 1986, and he is presently Professor of Computer Science from 1991, and Head of the Computer Science Department of UCM. His main research topic at present is “Formal Models of Concurrency”, and more specially the semantics of timed and probabilistic models.

Mr. Manuel Núñez graduated in “Mathematics (Computer Science)” in 1992 from the Universidad Complutense de Madrid (UCM) and is finishing his PhD in “Computer Science” with a Thesis devoted to “Testing Semantics for Probabilistic Process Algebras”. He is Assistant Teacher from 1992 at the Department of Computer Science of UCM. His main research area is “Formal Models of Concurrency”, and more specifically the semantics of probabilistic processes.

Prof. Juan Quemada graduated in “Telecommunications Engineering” in 1976 from the Universidad Politecnica de Madrid (UPM) and achieved a PhD in ”Telecommunications Engineering” in 1981 with a work on “Synchronization of Time Compression Multiplex Systems”. He was lecturer at the Faculty of Informatics before joining the Telematics Engineering Department of UPM in 1982 where he is presently Professor and Head of the Department. His present research interest is focused on the use of Formal Methods in the design of communication and distributed systems. He has given courses, conferences, authored over a hundred technical publications, including over 20 co-authored books. He is a regular member of scientific conferences such as FORTE or PSTV IFIP conferences.