Formal Aspects of Computing (1998) 10E: 28–96

Formal Aspects of Computing

A Formal Definition of Time in LOTOS

Luc Léonard and Guy Leduc

University of Liège, Institut Montefiore, Liège, Belgium

Keywords: LOTOS; ET-LOTOS; Real Time; Operational semantics; Negative premises

Abstract. Enhanced Timed-LOTOS, denoted ET-LOTOS, is an extension of LOTOS that allows the modelling of real-time behaviours. It covers all the aspects of full LOTOS, including data types, it supports both a dense and a discrete time domain and can manipulate time values as any other data values. A tutorial on ET-LOTOS, showing many application examples, has already been published [LéL97]. The present paper adds to it by providing an in-depth presentation of its theoretical aspects. The complete semantics is given and explained, and its properties are studied. In particular, we prove that the semantics is consistent and that strong bisimulation is a congruence. This requires to deal carefully with the presence of negative premises in the operational semantics, which are necessary to express urgency. ET-LOTOS is also shown to be a conservative extension of LOTOS for guarded processes, and is the basis of the timed extension of LOTOS currently developed by ISO [ISO98]. To our knowledge, this is the first in-depth study of a language that combines data types and real-time behaviours.

1. Introduction

The formal description technique LOTOS [BoB87, ISO8807] is an expressive specification language for concurrent and distributed systems. The flexibility of its operators generally allows clear and intuitive descriptions in various specification styles. However, the use of LOTOS has also revealed, or confirmed, some weak points of the language, and research is currently going on to overcome them. In the framework of ISO/IEC JTC1/SC21/WG7 a working group has been created for the definition of a new standard, called E-LOTOS (Extended LOTOS).

Among the desirable improvements to LOTOS, one of the most commonly admitted is the ability to specify adequately time-sensitive systems. Although LOTOS allows the description of systems by the temporal ordering of their actions,

it makes no provision for the expression of quantitative timing information. Because of this deficiency, one cannot fully specify time-sensitive systems. This can be shown easily with the very common case of a recovery mechanism based on time-outs.

The following process is a typical LOTOS specification of the classical time-out mechanism: send; (ack; SEND_NEXT □ i; RE_SEND). The internal event i is intended to model the expiration of the time-out and thus, the recovery mechanism. It is clear however that such a specification omits an important information, which is the value of the time-out. Simply adding this information as a parameter of the process would be insufficient too: it is also necessary to ensure that the ordering of events between concurrent systems be coherent with the specified time values.

Many other examples were available. More recently, new protocol mechanisms, as well as corresponding service facilities, have appeared, which strengthen the need for a "time extended" LOTOS. Isochronous data transfers, rate control and multimedia synchronization are some new examples.

In this paper, we present Enhanced Timed-LOTOS (ET-LOTOS for short), which is an extension of LOTOS with quantative time. On the one hand, ET-LOTOS offers very powerful features like the possibility to use of a dense time domain, or to manipulate time values like any other data values. On the other hand, the semantics of the formalism has been thoroughly studied. A sign of the interest of the work we present here is that ET-LOTOS has been retained as the basis of the timed extension of LOTOS currently developed in the ISO/IEC JTC1/SC33 standardization committee [ISO98].

ET-LOTOS is an extended and improved version of [LeL93, LéL94]. In particular, the language presented in [LéL94] has been enhanced and slightly modified to cover full LOTOS (i.e. LOTOS with data types). Among the other proposals that have inspired us, let us mention Real Time CSP [ReR88], TCCS [MoT90] and the Calculus of Real-Time Systems [Wan91]. A detailed comparison with these proposals, and others, is provided in Section 8.

In this paper we present ET-LOTOS from a theoretical point of view. Its expressive power as a specification language is further demonstrated in another paper [LéL97] where the language is used to model many relevant systems.

2. Overview of the Paper

In standard LOTOS, a system can only change state by performing actions, either internally or by interacting with its environment. Since the aim of ET-LOTOS is to allow the description of the influence of time on the systems, a second dimension of evolution is introduced: not only does the state of a system change when an event occurs, but also simply because of the passing of time. In ET-LOTOS the traditional action transitions of the operational semantics of LOTOS are thus complemented by timed transitions. Intuitively, $P \xrightarrow{a} P'$ means that in the state described by process P, the system can perform action a and then reach the state described by P'. Similarly, $P \xrightarrow{d} P'$, where d is a time value, means that a system initially in the state P will be in the state P' d time units later, provided that no action occurs in the meantime. Note that no difference is made in the denotation between the action and the time transitions. The distinction is made by the nature of the label of the transition. In ET-LOTOS, the classical alphabet of actions of LOTOS, A, is extended with D, called the time domain, which is the set of all possible time values.

The first step in the presentation of ET-LOTOS is thus the definition of its time domain in Section 3. Time values will be considered as any other data values in ET-LOTOS, so that the specifier is free to specify the time domain. He simply has to respect some requirements linked to the semantic model of ET-LOTOS.

In Section 4 we present the syntax of our language.

Then we illustrate the expressive power of ET-LOTOS on a token bucket algorithm in Section 5. The operational semantics is given in Section 6. We first present the model used to describe the behaviour of ET-LOTOS processes. As in LOTOS, it is a Labelled Transition Systems (LTS). To associate each process with its LTS, we provide a Transition Derivation System (TDS), i.e. for every operator, we give a set of axioms and inference rules defined in the classical Plotkin's style [Plo81]. Then we explain how an ET-LOTOS process is associated with its LTS. The procedure is classical but the use of negative premises in the TDS raises some difficulties. We discuss them and show how to overcome them in our model.

Section 7 is dedicated to a study of the properties of our model. The strong bisimulation equivalence is extended to cope with time transitions, and is proved to be a congruence. We also develop equivalence laws, and we discuss the compatibility between the ET-LOTOS and LOTOS theories. We also point out some unrealistic behaviours which can be specified with ET-LOTOS. Some are indeed useful abstractions, but others are simply considered as badly defined processes. However, it is worth being aware of their existence.

Finally, Section 8 is a comparison between ET-LOTOS and other existing timed formalisms. This is also the place where we discuss the problems raised by the upgrade from basic ET-LOTOS [LéL94] to full ET-LOTOS.

More details can be found in [Léo97].

3. The Time Domain

An important feature of ET-LOTOS is that time values are considered like any other data. Thus, they need to be defined in the LOTOS Abstract Data Type language. We still assume that this formalism is ACT ONE [MRV92], even though much work is carried out to provide the new LOTOS standard, E-LOTOS, with a language based on a more operational model [ISO98]. A very brief introduction to the semantics of ACT ONE and to the terms and notations we use in the sequel is given in Annex A2.

In the sequel, we will denote by D the time domain. Assuming that the time values are of sort *time*, $D = \Theta(time)$, where $\Theta(s)$ means the set of values of sort s.

In fact, the language used for the definition of the data types does not really matter. The semantic model of ET-LOTOS simply requires that some semantical elements be defined on D, the time domain. Provided that this constraint is respected, the specifier is left free to define his own time domain.

Variables d, d', $d''\ldots, d_1$, $d_2\ldots$ are taken to range over D. $D_\infty = D/\{\infty\}$, $D_{0\infty} = D/\{0, \infty\}$.

So, the following elements have to be defined on D[1]:

- A total order relation represented by "$<$".
- An element $0 \in D$ such that: $\forall\, d \in D \bullet d \neq 0 \Rightarrow 0 < d$
- An element $\infty \in D$ such that: $\forall\, d \in D \bullet d \neq \infty \Rightarrow d < \infty$

[1] This formal definition was inspired by the one given in [MFV94].

- A commutative and associative operation " $+ : D, D \to D''$ such that:

$$\forall d, d' \in D \bullet d < d' \Leftrightarrow \exists d'' \in D \bullet d'' \neq 0 \wedge (d + d'') = d'$$

that is, time always passes: $0 < d \Rightarrow d' < d' + d$
and the time interval between two time instants can always be measured

$$\forall d \in D \bullet d + 0 = d$$
$$\forall d \in D \bullet d + \infty = \infty$$

The relations " $\leqslant$ " and " $-$ " can be derived easily as follows:

$$\forall d, d' \in D \bullet d \leqslant d' \Leftrightarrow (d < d' \vee d = d')$$
$$\forall d, d'' \in D, d' \in D_\infty \bullet d - d' = d'' \Leftrightarrow d' + d'' = d$$

We say that D is:

$$\textit{dense} \text{ iff } \forall d, d' \in D \bullet \ (d < d' \Rightarrow \exists d'' \in D \bullet \ d < d'' < d')$$
$$\textit{discrete} \text{ iff } \forall d \in D_\infty \exists d' \in D_\infty \bullet (d < d' \wedge \forall d'' \in D \bullet d'' \leqslant d \vee d'' \geqslant d')$$

Possible time domains are e.g. $\mathbb{N} \cup \{\infty\}$, $\mathbb{Q}^{\geqslant 0} \cup \{\infty\}$, $\mathbb{R}^{\geqslant 0} \cup \{\infty\}$. However, notice that ACT ONE only allows the definition of countable data types, i.e. a time domain isomorphic to $\mathbb{R}^{\geqslant 0} \cup \{\infty\}$ could not be specified.

The time domain can also consist of just one element standing for 0 and ∞. In this case, ET-LOTOS amounts to an untimed formalism, since no time transition is possible.

4. The Syntax of ET-LOTOS

Like any LOTOS specification, an ET-LOTOS specification is made of two parts: the definition of the data types and the definition of the behaviour. The data types part is the same as in LOTOS, but it must contain the definition of the time domain, following the constraints specified in the previous section. The syntax of the behaviour part is given below.

4.1. Notations

S and V denote respectively the sets of sorts and of data values in the initial algebra associated with the data types specification: $V = \bigcup_{s \in S} \Theta(s)$.

G is the finite set of common observable gates and $OG = G \cup \{\delta\}$ is the alphabet of observable gates, where δ is the special action denoting successful termination ($\delta \notin G$). δ does not appear explicitly in the syntax of ET-LOTOS.

$OA = OG \times V^*$ denotes the set of observable actions. $A = OA \cup \{\mathbf{i}\}$ denotes the alphabet of actions, where the symbol $\mathbf{i}$ is reserved for the unobservable internal action ($\mathbf{i} \notin OG$).

The variables g, a ($b, c, \ldots$ except g and d), α and v respectively range over G, OA, A and $A \cup D_{0\infty}$, $gv_1 \ldots v_n$ and $\delta v_1 \ldots v_n$ denote elements of OA, with the $v_i' s \in V$. Γ will be used to denote subsets of G. The function $\underline{\text{name}}: A \to OG \cup \{\mathbf{i}\}$ indicates the gate where an action occurs: $\underline{\text{name}}(gv_1 \ldots v_n) = g$, $\underline{\text{name}}(\delta v_1 \ldots v_n) = \delta$, $\underline{\text{name}}(\mathbf{i}) = \mathbf{i}$.

P, P', P'', $Q \ldots$ denote ET-LOTOS processes.

$[t + d/t]SP$ and $[t + d/t]P$ denote the selection predicate SP and the process P where the variable t has been syntactically replaced by $t + d$.

4.2. Abstract Syntax

The abstract syntax of the core ET-LOTOS language is defined by the following BNF expressions, where the new features are outlined. Some shorthands are also added for convenience. An informal presentation of the LOTOS operators is given in annex A1. Therefore, we will only explain their time extensions. In these expressions, N is a process identifier, the g_i's are gate names, the tx_i's are terms, td is a term of sort *time*, the e_i's represent a term tx[2], the o_i's represent either $?x{:}s$ (with x a variable of sort s) or $!tx$, the x_i's are variables of sorts s_i's and in $@t$, t is a variable of sort *time*.

$P ::= B$ **where** $\tilde{X} := \tilde{B}$[3]

$B ::=$	**stop**	(inaction, idling)
	| **block**	(timelock)
	| **exit**$(e_1, \ldots e_n)(\{td\}$	(successful termination)
	| $go_1 \ldots o_n @t[SP]; B$	(observable action-prefix)
	| $\mathbf{i}@t\{td\}; B$	(internal action-prefix)
	| $\Delta^{td} B$	(delay prefixing)
	| $B \Box B$	(choice)
	| $B \lvert[\Gamma]\rvert B$	(parallel composition)
	| **hide** Γ **in** B	(hiding)
	| $B >>$ **accept** $x_1{:}s_1, \ldots x_n{:}s_n$ **in** B	(enabling)
	| $B [> B$	(disabling)
	| $[SP] \rightarrow B$	(guard)
	| **let** $x_1 = tx_1, \ldots x_n = tx_n$ **in** B	(instantiation)
	| **choice** $x_1{:}s_1, \ldots x_n{:}s_n \Box B$	(choice over values)
	| **inf** $\lvert\lvert\lvert B$	(infinite parallel composition)
	| $N[g_1, \ldots g_n](tx_1, \ldots tx_n)$	(process variable)

$X ::= N[g_1, \ldots g_n](x_1{:}s_1, \ldots x_n{:}s_n)$

Shorthand notations: We keep the classical LOTOS notations for the parallel composition operator: $P \parallel P$ stands for $P \lvert[G]\rvert P$ and $P \lvert\lvert\lvert P$ for $P \lvert[\varnothing]\rvert P$.

The informal meaning of the new operators and constructs are as follows:

- **block** is a process that cannot perform any action, nor age. It models a pathological process that freezes time. It is not supposed to be used directly in specifications, but can results from incompatible timing constraints. Examples of processes that behave like **block** will be given in Section 7.5.2.
- In **exit**$(e_1, \ldots e_n)\{td\}$, the successful termination can occur between 0 and td time units only. Beyond td, the process turns into **stop**.
- In $go_1 \ldots o_n @t[SP]; P$, the declared variable t (of sort *time*) is used to measure the elapsed time between action offer (along gate g) and its actual occurrence. It can appear in the selection predicate SP (and thus constrain the possible occurrence times) and in the subsequent behaviour P.
- In $\mathbf{i}@t\{td\}; P$, the purpose of the declared variable t is as above, and the life-reducer $\{td\}$ means $[0 \leqslant t \leqslant td]$. The internal action **i** will necessarily occur in this interval, but at an unpredictable time. This feature allows the modelling of nondeterministic delays.

---

[2] e_i can also be "any s" (with $s \in S$), to denote any term of sort S.

[3] For convenience, we suppose, without lack of generality, that there is a single where-clause that gathers all the process declarations of the specification.

- $\Delta^{td}\ B$ is a process that behaves like B after having waited for td time units. This operator introduces a fixed delay.
- **inf** ||| B is a process that behaves like an infinite number of B's running in parallel without synchronisation.

In ET-LOTOS, observable actions may only occur when the associated selection predicate is true, but there is no obligation for an observable action to occur, even at the last time allowed by the predicate (if any). One usually says that observable actions are not urgent, although this terminology can be misleading. Perhaps a better informal meaning could be to say that they cannot be subject to deadlines. This is in line with the intuitive understanding that an observable action is constrained by the external environment: its occurrence is a rendezvous interaction with some other process(es) which may not be ready to execute the action at any time. Therefore, enforcing the occurrence of an observable action would mean that a process has full control on its environment. This would be counter-intuitive and not compositional. For example, consider a process P with a local time deadline on an observable action g. If one verifies the timing properties in isolation, P will satisfy some timed liveness properties with respect to g. Now put P in a parallel context such that it synchronizes with a process Q that does not offer g within the proposed timed window of P. Clearly, g cannot occur. This means that the timing property verified by P is lost. Therefore, there are only two consistent solutions: either we keep the classical synchronized parallel operator and observable actions should be non urgent, or we define a new synchronization operator which is compatible with urgent observable actions. For compatibility purposes, we have chosen the first solution, but we will come back to the second one when discussing other models in Section 8.

On the other hand, internal actions are urgent when their associated life-reducer expires. This is reasonable because an internal action is completely autonomous, i.e. it cannot be constrained by the environment.

Now, to get enough expressive power without observable actions being urgent, we have to add another feature in the language, called *maximal progress*. When two or more processes synchronize through an observable gate g, they usually put different timing constraints on the possible occurrence times of g. The interaction along g, which is still observable, and thus non urgent, is therefore restricted to occur when all the timing constraints are satisfied. In ET-LOTOS, such an (inter)action can only become urgent (i.e. occur at the first time that satisfies all the constraints) when we can make sure that no additional process (say the environment) can further take part in it, otherwise the first time might be postponed. By definition, to isolate an (inter)action from its environment, the only way is to hide it, i.e. to turn it into an internal action. Then the *maximal progress* property of hidden actions (captured by semantic rule H3 in the sequel) will ensure that hidden actions or interactions among processes occur as soon as possible, viz. as soon as allowed by all the processes involved. This simple notion of urgency turns out to be very expressive in practice [LéL97].

Additional shorthand notations

In simple cases the selection predicate can be replaced by a life-reducer:

$go_1 \ldots o_n\{d\}; P = go_1 \ldots o_n@t[t \leqslant d]; P$ provided t is not a free variable in P.

or by an interval:

$go_1 \ldots o_n\{d, d'\}; P = go_1 \ldots o_n@t[d \leqslant t \leqslant d']; P$ provided t is not a free variable in P.

An interval can also be used with internal actions:

$\mathbf{i}\{d, d+d'\}; P = \Delta^d\mathbf{i}\{d'\}; P$
$\mathbf{i}@t\{d, d+d'\}; P = \Delta^d\mathbf{i}@t\{d'\}; [t+d/t]P.$

Remark:

- In $go_1 \ldots o_n@t[SP]; P$ we let both $@t$ and $[SP]$ be optional, and use the convention that, if omitted, $[SP] = [true]$. In $\mathbf{i}@t\{d\}; P$, both $@t$ and $\{d\}$ are optional. If omitted, $d = 0$. Similarly $\{d\}$ is optional in **exit**$\{d\}$, and **exit** means implicitly **exit**$\{\infty\}$.
- In the sequel we use indiscriminately the terms "behaviour expression" and "process". It is not necessary here to make a distinction, even though, strictly speaking, the behaviour of a process is defined by a behaviour expression.
- Process definitions can be recursive. When an instance of a process X appears in a behaviour expression, this means the instantiation of the process definition associated with X.
- These operators have the following decreasing binding power: action and delay prefixing, choice, parallelism, disabling, enabling, hiding, infinite parallelism.
- Processes are considered equal modulo equality of the data types, e.g.: **exit**$\{1+1\} =$ **exit**$\{2\}$.

5. Specification Example – A Token Bucket Algorithm

To give an example of the expressing capabilities of ET-LOTOS, we apply it to a realistic example: the interface between a "token bucket" congestion control algorithm and a network. It illustrates how various mechanisms can be combined with each other in ET-LOTOS thanks to the maximal progress on hidden actions. It also illustrates how complex mechanisms can be specified easily by the combined use of $@t$ and selection predicates.

We take the following explanations about the token bucket algorithm from [Tan96].

One of the main causes of congestion in networks is that traffic often comes in bursts. A widely used method to manage this problem in ATM networks is called "traffic shaping". It is about regulating the average transmission rate (and burstiness) of a host on the network. Conceptually, each host is connected to the network by an interface containing a finite queue, called the "bucket".

If a packet arrives at the queue when it is full, the packet is discarded. Various algorithms exist regarding the output policy of the queue. The one that we describe here, called "token bucket", allows the output to speed up somewhat when large bursts arrive. In this algorithm, the bucket holds tokens generated by a clock at the rate of one token every T sec. Each token represents the right to send α bytes. A packet can only be transmitted if enough token are available to cover its length in bytes. Fractions of tokens are kept for future use.

The token bucket algorithm is not the only factor determining the transmission time of a packet. Another element is the throughput of the network. The transmission of a packet is indeed not instantaneous. It takes a time equal to the

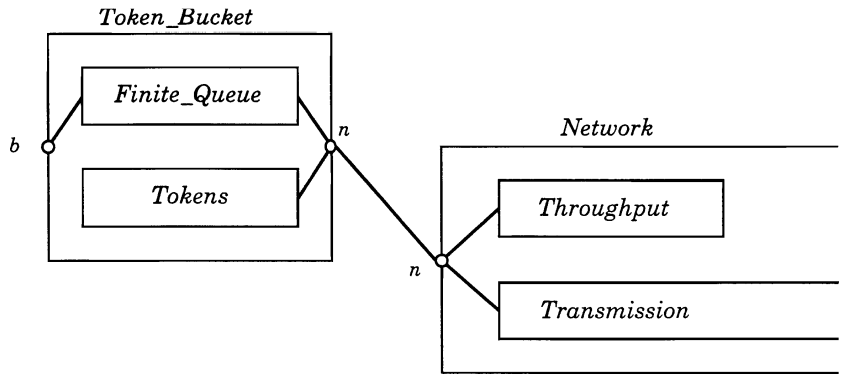

Fig. 1. The Token Bucket and the network.

```
Token_Bucket [b, n] :=
Finite_Queue[b, n] (queue:FifoQueue)
|[n]|
Tokens [n] (max_size:Nat,t1,period:time)
```

Fig. 2. The Token Bucket process.

```
Finite_Queue[b, n] (queue:FifoQueue) :=
b?p:Packet [size(p) ≤ room_in(queue)] ; Finite_Queue[b, n] (Append(p,queue))
□ b?p:Packet [size(p) > room_in(queue)] ; Finite_Queue[b, n] (queue)
□ [not(IsEmpty(queue))]→ n!TopOf(queue) ; Finite_Queue[b, n] (Cut(queue))
```

Fig. 3. Finite FIFO Queue process.

```
Tokens[n](max_size: Nat,t1,T:time) :=
    (* max_size is the maximal size number of bytes that can be transmitted (at instantiation time);
       t1 is the time elpased since the last token arrival;
       T gives the time between token arrivals *)
n?p:packet@t [size_of(p) ≤ max_size + α * ((t+t1) div T)];
Tokens[n](max_size + α * ((t+t1) div T) - size_of(p), (t+t1) rem T, T)
```

Fig. 4. The output algorithm.

ratio between the size of the packet and the throughput offered by the network (which for simplicity, we will assume to be constant). Hence, even if enough tokens are available, two packets cannot be transmitted at the same time.

This system is shown on Fig. 1. The *b* gate is between the host and its token bucket interface, the *n* gate is between the token bucket and the network. It is assumed that these gates are hidden in the complete specification.

Token_Bucket is composed of two processes (Figs 1 and 2): *Finite_Queue* and *Tokens*.

Finite _Queue is very simple. It just uses a FIFO queue to ensure that all the cells be delivered in their transmission order. A packet received from the host is added to the queue only if enough room remains in the queue, otherwise it is discarded.

The interesting point is the specification of *Tokens*. The goal of this process (Fig. 4) is to express the effect of the output algorithm explained above. Hence, the packet at the top of the queue may be transmitted only when enough tokens have been generated to cover its size. Many solutions were possible to describe this mechanism. The one we present takes maximal advantage of data types. It illustrates the expressive power of the action-prefix extended with the @t construct

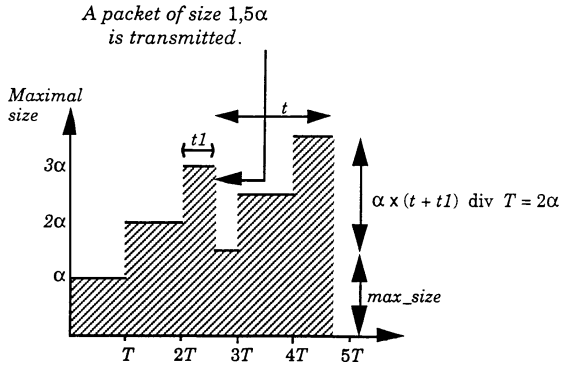

Fig. 5. Behaviour of the Tokens process.

$Throughput[n] := n?p{:}packet;\ \Delta^{size\_of(p)/\beta}\ Throughput[n]$

Fig. 6. Throughput of the network.

and associated with the selection predicate. The line "$n?p{:}packet@t\ [size\_of(p) \leqslant \ldots]$" expresses that an action along gate n (modelling the transmission of a packet p) can only occur at time t if the size of the packet is smaller than an expression that increases with t. In other words, this captures a rate control that allows bigger packets to be accepted after greater delays.

Figure 5 will help understand the meaning of the different parameters. It describes the behaviour of *Tokens* assuming that *max_size* equals α at time *0*. The values indicated are the ones just before time *5T*.

We use the mathematical operators divide (*div*) and remainder (*rem*) which we assume to be defined in the data types ($x = y \times (x\ div\ y) + (x\ rem\ y)$). Remember that one token permits to transmit α bytes and that a new token is generated every T sec. The function *size_of* gives the size in bytes of a packet.

There is no variable recording explicitly the number of tokens available. At any time, the maximal number of bytes which can be transmitted is given by the function: $max\_size + \alpha * ((t + t1)\ div\ T)$. In this expression, t records the time elapsed since the last occurrence of n, i.e. since the last transmission of a packet. The value of *t1* does not vary with the passing of time, but is updated, at each occurrence of n, by the time elapsed since the last addition of a token. Hence, at any time, $(t + t1)\ div\ T$ indicates the number of tokens added since the last transmission of a packet. *max_size* is also updated at each occurrence of n. It records the remaining number of bytes that can be transmitted.

We will not specify *Network* in details. Just the process *Throughput* is of interest to us. The remainder of the system is supposed to be captured in the process *Transmission* that we do not describe.

To describe *Throughput*, it is important to determine the actual meaning associated with the occurrence of an event on gate n. Does it mark the end or the beginning of the transmission of the packet? The constraint expressed by *Tokens* is indeed on the starting time of the transmission of a packet. Then, n indicates the beginning of the transmission and assuming a throughput of β bytes/sec and a packet of size s, this transmission lasts s/β sec. In other words, the network may not accept a new packet before s/β sec. This is what is expressed by *Throughput* (Fig. 6).

Notice that this example relies on the maximal progress hypothesis. The time at which a packet may be transmitted is the result of the composition of several

constraints, none of them having the control of the transmission alone. But the maximal progress on the occurrence of n ensures that each packet is transmitted as soon as possible.

6. The Semantics of ET-LOTOS

We provide the operational semantics of ET-LOTOS by translating every syntactically correct process into a Labelled Transition System. We explain below how the passing of time is described in this model. Then we detail the Transition Derivation System of our model which serves to link each process with its LTS. It is defined as a set of axioms and inference rules in the classical Plotkin's style.

6.1. "Timed" LTS

A Labelled Transition System (LTS) is a 4-tuple $\langle St, La, Tr, s_0 \rangle$, where St is a non-empty set of states, La is a set of labels, $Tr \subseteq St \times La \times St$ is a set of transitions and $s_0 \in St$ is the initial state.

In LOTOS, La is the set of actions and each state in St corresponds to a (closed) process. A transition (P, α, P'), denoted $P \xrightarrow{\alpha} P'$ in the sequel, may be read as: the process P may perform the action α and thereby be transformed into the process P'. We will sometimes write that the process is in the state P'.

We keep this model for ET-LOTOS. Following an idea first introduced by Moller and Tofts [MoT90], we simply extend it with a new kind of transitions: timed transitions. More precisely, we extend the set A of actions with D, the time domain.

A transition labelled with a time value d, $P \xrightarrow{d} P'$, means then: the process P may idle for d time units[4] and thereby be transformed into the process P'. Notice that no difference is made between action and time transitions in the denotation. They are distinguished only by the nature of their labels. Note also that timed transitions will only be labelled with time values in $D_{0\infty}$, i.e. ET-LOTOS processes do not perform time transitions of zero or infinite duration.

We will also use the following notations: $P \xrightarrow{\alpha}\!\!\!\!\!\!/\;$, with $\alpha \in A$, means that there exists no process P' such that $P \xrightarrow{\alpha} P'$, i.e. P cannot perform action α. $P \xrightarrow{g}\!\!\!\!\!\!/\;$, with $g \in G$, means that there exists no process P' and no action α with $\underline{\text{name}}(\alpha) = g$ such that $P \xrightarrow{g} P'$, i.e. P cannot perform any action along gate g. $P \xrightarrow{d}\!\!\!\!\!\!/\;$, with $d \in D_{0\infty}$, means that there exists no process P' such that $P \xrightarrow{d} P'$, i.e. P cannot idle during d time units.

Finally, we will sometimes use the shorthand notation $P_{(d)}$ to represent the behaviour expression defined by: $P \xrightarrow{d} P_{(d)}$ if $d > 0$ and $P_{(d)} = P$ if $d = 0$. Thanks to the time determinism property (see Section 7.1), if $P_{(d)}$ exists, it is unique. Hence, this notation is not ambiguous.

6.2. The Transition Derivation System

In the following inference rules: $d \in D_{0\infty}$, $d' \in D$, $d'' \in D_{\infty}$, $g \in G$ and $\alpha \in A$. We use a compact notation $\tilde{x}$ to denote an indexed set $(x_1, \ldots, x_n)$ and we write $\tilde{x}$ *op* $\tilde{y}$ for $(x_1 \; op \; y_1, \ldots, x_n \; op \; y_n)$, where *op* is either " / ", ":" or " = ".

[4] Depending on the time domain, d is not necessarily a natural number.

Notice that to be totally formal, the behaviour expressions should not contain time values in D but just terms of sort *time*. However, to avoid cumbersome notations, we take the convention to represent ground terms of sort *time* by the corresponding time values, e.g. we will write $\Delta^d P$ instead of $\Delta^{td} P$ where $[td] = d$.

Inaction: stop

$$\textbf{stop} \xrightarrow{d} \textbf{stop} \qquad \text{(S)}$$

As in standard LOTOS, **stop** is just a process that may not perform any action. However, rule (S) indicates that **stop** may idle indefinitely.

Timelock: block

Block cannot perform any action, nor age. It has no axiom. It is a pathological process that is in deadlock and additionally freezes time, a so-called timelock. This process is not intended to be used directly in the language, but it is useful to have it in the process algebra in order to express incompatible timing constraints that can result from wrong specifications.

Delay prefixing: $\Delta^d P$

$$\text{(D1)}\quad \frac{P \xrightarrow{\alpha} P'}{\Delta^0 P \xrightarrow{\alpha} P'} \qquad \Delta^{d'+d} P \xrightarrow{d} \Delta^{d'} P \quad \text{(D2)}$$

$$\frac{P \xrightarrow{d} P'}{\Delta^{d''} P \xrightarrow{d+d''} P'} \quad \text{(D3)}$$

According to intuition, we see that $\Delta^d P$ may not perform any action unless the delay is elapsed ($d = 0$). (D2) indicates the effect of the passing of time before the delay elapses (i.e. equals *0*). (D3) completes (D2). It indicates that the expiration of a delay does not prevent a process from continuing to idle if it can.

Observable action-prefix: $go_1 \ldots o_n @ t[SP]; P$

$$\text{(AP1)}\quad g\tilde{o}@t[SP]; P \xrightarrow{g\ v} [\tilde{v}/\tilde{o}, 0/t]P \qquad g\tilde{o}@t[SP]; P \xrightarrow{d} g\tilde{o}@t[[t+d/t]SP]; [t+d/t]P \quad \text{(AP2)}$$

if
- for all i, $o_i = !tx$ implies $v_i = [tx]$
- for all i, $o_i = ?x{:}s$ implies $v_i \in \Theta(s)$
- $\vdash [\tilde{v}/\tilde{o}, 0/t]SP$

where $v_i/o_i = tx_i/x$ with $[tx_i] = v_i$ if $o_i = ?x{:}s$

v_i/o_i is void if $o_i = !tx$

The main interest of the above rules lies in the description of the mechanism used to measure the passing of time. For the rest, (AP1) is similar to the equivalent LOTOS rule and (AP2) simply indicates that a process prefixed by an observable action can progress in time indefinitely.

So, let us concentrate on $@t$. Notice first that the only possible instantiations of the attributes $\tilde{o}$ of g are the ones verifying the predicate SP at $t = 0$ (AP1). Instantiating t by *0* is in fact logical: $g\tilde{o}@t[SP]; P$ describes a process at a given time and the counting of t starts at that time. So, t is still at *0* if the process immediately performs action $g\tilde{o}$. The way the value of t is updated when the process idles is described by rule (AP2).

In (AP2), the only effect of the passing of time is indeed on the time measurement variable: after each time transition of d units, t is syntactically replaced, in P and in SP, by $t+D$. So, $g\tilde{o}@t[SP]; P$ turns into $g\tilde{o}@t[[t+d/t]SP]; [t+d/t]P$. Hence

according to (AP1), the occurrence of $g\tilde{v}$ transforms the process into: $[\tilde{v}/\tilde{o}, 0+d/t]P$, the final result being that t is actually instantiated by d.

Internal action-prefix: $\mathbf{i}@t\{d\}; P$

$$(I1)\quad \mathbf{i}@t\{d'\}; P \xrightarrow{\mathbf{i}} [0/t]P \qquad\qquad \mathbf{i}@t\{d'+d\}; P \xrightarrow{d} \mathbf{i}@t\{d'\}; [t+d/t]P \quad (I2)$$

These rules are interesting because they describe how the notion of "necessity" is reflected in the semantics. As regards $@t$, the mechanism is the same as for observable actions.

Remember that $\{d\}$, the life reducer, expresses that **i** will occur within the next d time units. This amounts to saying that $\mathbf{i}\{d\}; P$ may not idle for more than d time units. This is indeed the effect of rule (I2): the only time transitions allowed are the ones with a value less than d. Hence, if **i** does not occur, the process eventually reaches the state $\mathbf{i}\{0\}; P$. And in this state, it cannot idle anymore: the progression of time is blocked. In this situation, **i** is said to be urgent. Note, however, that there is no risk of deadlock with an urgent **i**: its occurrence is always possible as it does not require an interaction with the environment.

By comparison, notice that $a\{d\}$ is a shorthand for $a@t[t \leqslant d]$. In this last case, we see by rule (AP2) that there is no restriction on the time transitions which the process may perform. But if $a@t[t \leqslant d]; P$ idles for more than d time units, let us say d'' time units with $d'' > d$, the resulting state is $a@t[t+d'' \leqslant d]; P$ and there is no way for the selection predicate to be satisfied again.

This way of enforcing urgency at the expiration of the life reducer is sometimes called a "must timing" policy, in contrast with the "may timing" policy of the observable action-prefix.

Exit: $\mathbf{exit}(e_1, \ldots e_n)\{d'\}$

$$(Ex1)\quad \mathbf{exit}(\tilde{e})\{d'\} \xrightarrow{\delta\tilde{v}} \mathbf{stop} \qquad\qquad \mathbf{exit}(\tilde{e}\,)\{d'+d\} \xrightarrow{d} \mathbf{exit}(\tilde{e})\{d'\} \quad (Ex2)$$

where $v_i = [t_i]$ if $e_i = t_i$ (a ground term)

$v_i \in \Theta(s_i)$ if $e_i = any\ s_i$

$$\mathbf{exit}(\tilde{e}\,)\{d'\} \xrightarrow{d} \mathbf{stop}\ (d > d') \quad (Ex3)$$

A life reducer can be associated with **exit**, meaning that the process may only terminate successfully within d' time units. Notice that this life reducer associated with **exit** follows a "may timing" policy. This is indeed logical because concurrent processes must synchronise on δ.

Choice: $P \square Q$

$$(Ch1)\quad \frac{P \xrightarrow{\alpha} P'}{P \square Q \xrightarrow{\alpha} P'} \qquad\qquad \frac{P \xrightarrow{d} P',\ Q \xrightarrow{d} Q'}{P \square Q \xrightarrow{d} P' \square Q'} \quad (Ch2)$$

$$(Ch1')\quad \frac{Q \xrightarrow{\alpha} Q'}{P \square Q \xrightarrow{\alpha} Q'}$$

Parallel Composition: $P \,|[\Gamma]|\, Q$

$$(PC1)\quad \frac{P \xrightarrow{\alpha} P'}{P \,|[\Gamma]|\, Q \xrightarrow{\alpha} P' \,|[\Gamma]|\, Q} \quad (\underline{\mathrm{name}}(\alpha) \notin \Gamma \cup \{\delta\}) \qquad \frac{P \xrightarrow{d} P',\ Q \xrightarrow{d} Q'}{P \,|[\Gamma]|\, Q \xrightarrow{d} P' \,|[\Gamma]|\, Q'} \quad (PC3)$$

$$(PC1')\quad \frac{Q \xrightarrow{\alpha} Q'}{P \,|[\Gamma]|\, Q \xrightarrow{\alpha} P \,|[\Gamma]|\, Q'} \quad (\underline{\mathrm{name}}(\alpha) \notin \Gamma \cup \{\delta\})$$

$$(PC2)\quad \frac{P \xrightarrow{\alpha} P',\ Q \xrightarrow{\alpha} Q'}{P \,|[\Gamma]|\, Q \xrightarrow{\alpha} P' \,|[\Gamma]|\, Q'} \quad (\underline{\mathrm{name}}(\alpha) \in \Gamma \cup \{\delta\})$$

Disabling: $P [> Q$

$$\text{(Di1)}\ \frac{P \xrightarrow{\alpha} P'}{P [> Q \xrightarrow{\alpha} P' [> Q}\ (\underline{\text{name}}(\alpha) \neq \delta) \qquad \frac{P \xrightarrow{d} P', Q \xrightarrow{d} Q'}{P [> Q \xrightarrow{d} P' [> Q'}\ \text{(Di4)}$$

$$\text{(Di2)}\ \frac{Q \xrightarrow{\alpha} Q'}{P [> Q \xrightarrow{\alpha} Q'}$$

$$\text{(Di3)}\ \frac{P \xrightarrow{\alpha} P'}{P [> Q \xrightarrow{\alpha} P'}\ (\underline{\text{name}}(\alpha) = \delta)$$

Basically, these three operators behave in ET-LOTOS as in LOTOS. What is worth pointing out is that they combine two processes that are both "enabled", i.e. both can perform actions. In this case, the rule is that time passes at the same pace in both processes.

This implies that a time blockage in one of the two processes also blocks the other. In particular, a choice is not resolved by the passing of time. Remember that blocking the passing of time is precisely the mechanism used to ensure the necessity of the internal actions.

Infinite parallel composition: inf $|||$ P

$$\text{(IP1)}\ \frac{P \xrightarrow{\alpha} P'}{\textbf{inf} ||| P \xrightarrow{\alpha} P' ||| (\textbf{inf} ||| P)}\ (\underline{\text{name}}(\alpha) \neq \delta) \qquad \frac{P \xrightarrow{d} P'}{\textbf{inf} ||| P \xrightarrow{d} \textbf{inf} ||| P'}\ \text{(IP2)}$$

Intuitively **inf** $|||$ P expresses an infinite number of interleaved occurrences of P. In other words, **inf** $|||$ $P = P |||$ (**inf** $|||$ P). From this expression, we could think that **inf** $|||$ is a derived operator in ET-LOTOS, and that the following expression would describe the same behaviour: $Ps := P ||| Ps$. However, for reasons which will be apparent later (refer to Section 7.5.2), unguarded processes like Ps block time in ET-LOTOS, whereas **inf** $|||$ P does not. The semantics of **inf** $|||$ is based on the idea that the whole behaviour of **inf** $|||$ P (including time transitions) can be fully deduced from the behaviour of P. This is clearly expressed by rule (IP2). (IP1) shows that the instances of P which have already performed an event are removed from the scope of **inf** $|||$. This operator is similar to the ! operator of the π-calculus [MPW92] and was first introduced in [Led86].

Hiding: hide Γ **in** P

$$\text{(H1)}\ \frac{P \xrightarrow{\alpha} P'}{\textbf{hide}\ \Gamma\ \textbf{in}\ P \xrightarrow{\alpha} \textbf{hide}\ \Gamma\ \textbf{in}\ P'}\ (\underline{\text{name}}(\alpha) \notin \Gamma)$$

$$\text{(H2)}\ \frac{P \xrightarrow{\alpha} P'}{\textbf{hide}\ \Gamma\ \textbf{in}\ P \xrightarrow{\mathbf{i}} \textbf{hide}\ \Gamma\ \textbf{in}\ P'}\ (\underline{\text{name}}(\alpha) \in \Gamma)$$

$$\frac{P \xrightarrow{d} P',\ \forall g \in \Gamma \bullet P \not\xrightarrow{g},\ (\forall g \in \Gamma\ \forall d'' < d \bullet P \xrightarrow{d''} \not\xrightarrow{g})}{\textbf{hide}\ \Gamma\ \textbf{in}\ P \xrightarrow{d} \textbf{hide}\ \Gamma\ \textbf{in}\ P'}\ \text{(H3)}^{5}$$

[5] $P \xrightarrow{d''} \not\xrightarrow{g}$ is just a shorthand for $P \xrightarrow{d''} P''$, $P'' \not\xrightarrow{g}$ for some P''. In particular, this P'' always exists when the first premise of H3 is true, and is unique. This results from the time additivity and time determinism properties to be presented in section 7.1. If we use the shorthand notation $P_{(d)}$ defined in Section 6.1 we can further simplify the premises of rule (H3) as follows: $P \xrightarrow{d} P_{(d)} \wedge \forall g \in \Gamma \forall d' < d \circ P_{(d')} \not\xrightarrow{g}$.

hide is associated with an important feature of the language: the maximal progress on hidden actions. This property states that any hidden event is urgent, i.e. that it must occur as soon as possible. This property is of course expressed in the semantics of **hide**.

The rules (H1) and (H2) are simply the LOTOS ones. (H3) is more interesting. The first premise is obvious: **hide** Γ **in** P may age only if P does, but this is not sufficient.

Consider for example the process **hide** g **in** $\Delta^3 g$; **stop**. At time *3*, g is enabled. Hence it must occur as soon as possible. The principle is the same as with an urgent **i**: at time *3*, the passing of time is blocked and the only remaining possibility is the occurrence of the hidden g. This means that we must not let **hide** g **in** $\Delta^3 g$; **stop** idle for more than *3* time units, although $\Delta^3 g$; **stop** can idle indefinitely. This is the role of the second and third premises to prevent the process from idling beyond the occurrence time of a hidden action.

Enabling: P >> accept $x_1{:}s_1, \ldots x_n{:}s_n$ **in** Q

$$\text{(En1)}\quad \frac{P \xrightarrow{\alpha} P'}{P \gg \textbf{accept } \tilde{x}{:}\tilde{s} \textbf{ in } Q \xrightarrow{\alpha} P' \gg \textbf{accept } \tilde{x}{:}\tilde{s} \textbf{ in } Q} \quad (\underline{\text{name}}(\alpha) \neq \delta)$$

$$\text{(En2)}\quad \frac{P \xrightarrow{\delta \tilde{v}} P'}{P \gg \textbf{accept } \tilde{x}{:}\tilde{s} \textbf{ in } Q \xrightarrow{\mathbf{i}} [\tilde{t}x/\tilde{x}]Q} \quad \text{where } \forall i \bullet [tx_i] = v_i \in \Theta(s_i)$$

$$\frac{P \xrightarrow{d} P',\ P \overset{\delta}{\not\rightarrow},\ (\forall d'' < d \bullet P \xrightarrow{d''} \overset{\delta}{\not\rightarrow})}{P \gg \textbf{accept } \tilde{x}{:}\tilde{s} \textbf{ in } Q \xrightarrow{d} P' \gg \textbf{accept } \tilde{x}{:}\tilde{s} \textbf{ in } Q} \quad \text{(En3)}^6$$

Notice that the enabling operator is also concerned with the maximal progress on hidden events, as it hides the termination event. The mechanism is the same as for **hide**.

Guard: $[Sp] \rightarrow P$

$$\text{(G1)}\quad \frac{P \xrightarrow{\alpha} P'}{[SP] \rightarrow P \xrightarrow{\alpha} P'} \quad \text{if} \vdash SP \qquad \frac{P \xrightarrow{d} P'}{[SP] \rightarrow P \xrightarrow{d} P'} \quad \text{if} \vdash SP \quad \text{(G2)}$$

$$[SP] \rightarrow P \xrightarrow{d} \textbf{stop} \quad \text{if} \neg \vdash SP \text{(G3)}$$

Let: let $x_1 = tx_1, \ldots x_n = tx_n$ **in** Q

$$\text{(L1)}\quad \frac{[\tilde{t}\tilde{x}/\tilde{x}]\,P \xrightarrow{\alpha} P'}{\textbf{let } [\tilde{t}\tilde{x}=\tilde{x}] \textbf{ in } P \xrightarrow{\alpha} P'} \qquad \frac{[\tilde{t}\tilde{x}/\tilde{x}]\,P \xrightarrow{d} P'}{\textbf{let } [\tilde{t}\tilde{x}=\tilde{x}] \textbf{ in } P \xrightarrow{d} P'} \quad \text{(L2)}$$

Process instantiation: $Q[h_1, \ldots h_n](x_1{:}s_1, \ldots x_n{:}s_n) := P$

$$\text{(In1)}\quad \frac{[\tilde{g}/\tilde{h},\ \tilde{t}\tilde{x}/\tilde{x}]P \xrightarrow{\alpha} P',\ Q[\tilde{h}](\tilde{x}) := P}{Q[\tilde{g}](\tilde{t}\tilde{x}) \xrightarrow{\alpha} P'} \qquad \frac{[\tilde{g}/\tilde{h},\ \tilde{t}\tilde{x}/\tilde{x}]P \xrightarrow{d} P',\ Q[\tilde{h}](\tilde{x}) := P}{Q[\tilde{g}](\tilde{t}\tilde{x}) \xrightarrow{d} P'} \quad \text{(In2)}$$

There is not much to say about these processes whose behaviour is the same as in LOTOS. Just notice that the guard $[SP] \rightarrow$ is evaluated at time zero. If SP cannot

[6] $P \xrightarrow{d''} \overset{\delta}{\not\rightarrow}$ is again a shorthand for $P \xrightarrow{d''} P''$, $P'' \overset{\delta}{\not\rightarrow}$ for some P''.

be satisfied, $[SP] \rightarrow P$ behaves exactly like **stop**, i.e. it cannot perform any action and lets time pass.

Choice over values: choice $x_1{:}s_1, \ldots x_n{:}s_n \Box P =$ **Achoice**(0) $\tilde{x}{:}\tilde{s} \Box P$

$$\text{(GC1)}\quad \frac{[\tilde{tx}/\tilde{x}]P \xrightarrow{\alpha} P'}{\textbf{Achoice}(0)\ \tilde{x}{:}\tilde{s} \Box P \xrightarrow{\alpha} P'} \qquad \frac{[\tilde{tx}/\tilde{x}]P \xrightarrow{d+d''} \ \forall\ \tilde{tx} \bullet [tx_i] \in \Theta(s_i),\ i = 1,\ldots n}{\textbf{Achoice}(d'')\ \tilde{x}{:}\tilde{s} \Box P \xrightarrow{d} \textbf{Achoice}(d+d'')\ \tilde{x}{:}\tilde{s} \Box P} \quad \text{(GC3)}$$

$$\text{(GC2)}\quad \frac{[\tilde{tx}/\tilde{x}]P \xrightarrow{d} P'',\ P'' \xrightarrow{\alpha} P'}{\textbf{Achoice}(d)\ \tilde{x}{:}\tilde{s} \Box P \xrightarrow{\alpha} P'} \qquad \text{where the } tx_i \text{ are ground terms with } [tx_i] \in \Theta(s_i)$$

We finish our review of the ET-LOTOS operators with the choice over values. It owes this honour (or disgrace?) to its admirable fidelity to the untimed semantics. Although its behaviour introduces no new time-related feature, it was indeed and by far the most difficult operator to provide with a timed formal semantics.

However, the intuition we want to express is simple: the occurrence of an action resolves the choice, but the passage of time does not. But determining whether a choice over values can age requires to determine whether each of the possible instantiations of the choice can do so. Furthermore, and this is the main difficulty, it also requires to determine the result of the passing of time for each of these instantiations.

To get an idea of the problem, consider the process below:

choice $x{:}time \Box P(x)$ **where** $P(x) := \Delta^x g@t; Q \Box$ **exit**$\{2x\}$.

What is the result of this process after idling for d time units? It is hard to say because the evolution of $P(x)$ depends in fact on the value of x:

$$\text{When} \quad x \geqslant d: \quad P(x) \xrightarrow{d} \Delta^{x-d} g@t; Q \Box \textbf{exit}\{2x-d\}$$

$$x < d \leqslant 2x: \quad P(x) \xrightarrow{d} g@t; [t+d-x/t]Q \Box \textbf{exit}\{2x-d\}$$

$$2x < d: \quad P(x) \xrightarrow{d} g@t; [t+d-x/t]Q.$$

How is it possible then to describe in a single expression the process resulting from **choice** $x{:}time \Box P(x) \xrightarrow{d}$?

To solve this problem, we have introduced the auxiliary operator **Achoice**(d), where $d \in D_\infty$. By definition, **choice** $x_1{:}s_1, \ldots x_n{:}s_n \Box P =$ **Achoice**(0) $x_1{:}s_1, \ldots x_n{:}s_n \Box P$. **Achoice**$(d)$ somewhat postpones the difficulty: rule (GC3) shows that d is used to record the passing of time. The elapsed time is then taken into account when one tries to determine the possible transitions (GC2).

Although correctly defined, a problem remains with rule (GC3): its premise is infinite when $\Theta(s_i)$ is so for some i. This can be difficult to handle in practice, especially for tools. However, this problem is intrinsic to the **choice over values** operator and is faced, under a form or another, with any time extension of full LOTOS. Nevertheless, notice that unlike other extensions of LOTOS discussed in Section 8, ET-LOTOS does not need the **choice over values** to describe time mechanisms, thanks to the flexibility and the expressive power of its operators. Hence the variables defined in **choice over values** should usually have a limited influence on the ability of the process to age.

6.3. Mapping of an ET-LOTOS Process to its LTS

6.3.1. Introduction

So far, we have presented the axioms and inference rules that compose the Transition Derivation System of ET-LOTOS. It remains to explain the procedure used to associate a process with its LTS.

In LOTOS, this procedure is rather complex. It involves three phases [ISO 8807]. We will only consider the third one here, which is the only one actually concerned with the building of the LTS. The first two phases are basically the same as in LOTOS. We just retain their result: the provision of a CLS (Canonical ET-LOTOS Specification).

In ET-LOTOS, as in LOTOS, a process is interpreted within the scope of this CLS. It consists of a tuple, composed of a CAS (Canonical Algebraic Specification) and of a CBS (Canonical Behaviour Specification). The CAS provides the semantics of the data types appearing in the process (and in particular, of the time domain). Hence it determines the sets S of all the sorts and OP of the operations on these sorts. The CAS also specifies E, a set of conditional equations, from which a derivation system can be generated. On the basis of this derivation system, the CAS is interpreted as a quotient term algebra. The CBS provides a set of process definitions, which are pairs consisting of a process name P and a behaviour expression B: $pdef = \langle P, B \rangle$. When a behaviour expression uses a process variable, its value is the one specified by the CBS.

The building of the LTS is quite simple in principle. Not all the transitions describe a possible behaviour. For example: **stop** $\xrightarrow{a}$ **exit** is clearly wrong. This is precisely the role of the TDS to determine which transitions are correct. Intuitively, a transition is correct iff it can be "proved" by the axioms and inference rules of the TDS, which we write: $\text{TDS} \vdash P \xrightarrow{v} P'$. We delay until the next section the explanation of what this proof actually consists of.

Then, the LTS of a process P is the 4-tuple $\langle st, A \cup D, Tr, s_0 \rangle$ where:

(i) $St = Der(P)$
$Der(P)$ is the set of derivatives of P, i.e. the smallest set satisfying
  (a) $P \in Der(P)$
  (b) if $P' \in Der(P)$ and $\text{TDS} \vdash P' \xrightarrow{v} P''$ for some v, then $P'' \in Der(P)$.
Intuitively, St is the set of states reachable from the initial state P.
(ii) A and D are respectively the set of actions and the time domain.
(iii) $Tr = \{(P, v, P') \mid P, P' \in S \text{ and } \text{TDS} \vdash P \xrightarrow{v} P'\}$.
(iv) $s_0 = P$.

6.3.2. Basic Concepts

It remains to define how transitions are proved from our TDS. For a TDS without negative premises, as in LOTOS, this is straightforward: it suffices to build a well-founded proof tree, using the rules of the TDS as inference rules. Then, we have: $\text{TDS} \vdash P \xrightarrow{v} P'$ iff such a tree exists for the transition $P \xrightarrow{v} P'$.

For a TDS with negative premises, like ours, these proof trees cannot be used. They can only show the presence of a transition. But, in order to prove the negative premises of some inference rules, the absence of a transition must also be proved. This incompatibility is not easily overcome.

A first notion we need to clarify is what we mean by "absence of a transition". Therefore, we will consider that the purpose of the TDS is not simply to allow us to prove transitions one by one, but more generally to define the set of all the "correct" transitions. This set is called the Transition Relation (TR) (in the sequel, we will use TDS and TR to make general statements about transition specification systems and transition relations and we will denote respectively by R and RE the TDS and the TR of ET-LOTOS[7]). Following Bloom, Istrail and Meyer [BIM95] this TR is required to agree with the TDS. Intuitively, this means that:

- Any transition in the TR is the conclusion of an inference rule of the semantics whose premises are true. In case of positive premises, this means that the required transitions also belong to the TR. In case of negative premises, this means that the corresponding transitions do not belong to the TR.
  For example: $a\{5\}$; **stop** ||| **exit** $\xrightarrow{a}$ **stop** ||| **exit** belongs to RE *only if* $a\{5\}$; **stop** $\xrightarrow{a}$ **stop** also belongs to RE.
- When the premises of a rule are true, the conclusion necessarily belongs to the TR.
  For example: if $a\{5\}$; **stop** $\xrightarrow{a}$ **stop** belongs to RE then, by rule (PC1), $a\{5\}$; **stop** ||| **exit** $\xrightarrow{a}$ **stop** ||| **exit** also belongs to RE (as well as all the other possible conclusions).

These requirements are indeed very logical. In terms of logic programming, this means that the transition relation is a supported model of the TDS.

However, this notion of agreement just defines a general target. It is not operational in the sense that it does not explain how the TR is obtained from the TDS.

In [Gro90b, Gro93], J. F. Groote mentions two potential problems. The first risk is that there is no TR agreeing with the TDS. Negative premises can lead to inconsistencies of the form: a transition belongs to the TR iff it does not belong to it. Imagine for example a TDS consisting of this single rule: $\dfrac{P \overset{a}{\not\rightarrow}}{P \xrightarrow{a}$ **stop**$}$

To belong to the TR, the transition $P \xrightarrow{a}$ **stop** must not belong to it. But if it does not belong to the TR, then the premise of the rule is true and $P \xrightarrow{a}$ **stop** must belong to the TR. Clearly, no TR exists which agrees with this TDS.

The second problem concerns the uniqueness of the TR. Maybe different TRs exist which all agree with a given TDS. Hence, proving the consistency of the TDS is not sufficient: one must also indicate unambiguously how to select one of the possible TRs as a basis for the semantics.

6.3.3. Defining RE

In this section, we show how to derive RE from R. Part of the proof that RE actually agrees with R is also presented (the remainder of the proof is given in Annex A3).

Notice that in [Gro90b, Gro93], a method is proposed to prove consistency of a TDS and to associate a transition relation with it. It is based on the definition of a stratification of the TDS. Basically, in the context of timed process algebras, the principle is the following. It consists of checking that the TDS verifies two properties:

---

[7] Hence, when we write $P \xrightarrow{x} Q$, this is to be understood as $P \xrightarrow{x} Q \in RE$. Similarly, $P \overset{x}{\not\rightarrow}$ means formally: $\forall Q \bullet P \xrightarrow{x} Q \notin RE$.

(1) the inference rules used to derive *action* transitions only have *positive action* transitions as premises;
(2) all the negative premises appearing in the inference rules used to derive *time* transitions are negative premises on *action* transitions.

For then, the derivation of the transition relation can easily be done in two stages. Firstly, one derives the set of all the action transitions (the first stratum). This is straightforward as one does not have to cope with negative premises or time transitions. Then, one derives the set of all the time transitions (the second stratum). As the negative premises are on action transitions only, it suffices to check in the first stratum whether they are verified or not.

In our case however, R verifies the second property, but not the first one. This is due to rule (GC3), where a time transition is used as premise to derive an action transition. Other kinds of stratifications have been searched, but none were found.

Hopefully, the existence of a stratification is not a necessary condition. In [BG96] a reduction method is proposed, by which the semantics can be stepwise reduced until it is either trivially consistent (i.e. has only positive premises) or can be stratified. If this can be done, the initial semantics is also consistent and strong bisimulation is a congruence. However the reduction steps are extremely complex due to the complexity of our language. Moreover, all our attempts to use this method quickly turned out to require a vast set of annex proofs and lemmas, which clearly lowered the interest of pursuing in this direction. Even for simpler process algebras, this method requires clever annex proofs and lemmas. Finally, the method is not sufficient to prove that strong bisimulation is a congruence in recursion contexts.

Another method is proposed by Baeten & Verhoef [BV93, Ver95]. It does not propose a solution to the consistency problem, but only congruence theorems. It would also require to rewrite the semantics to adhere to the path and panth formats of the method. These formats preclude negative premises which have to be rewritten as positive predicates, and the method still requires to find a stratification. Therefore, one can wonder whether the use of predicates could allow us to find an equivalent, but stratifiable, definition of our SOS. In fact, we have already explored this possibility in early versions of our language and we have abandoned it. Either we define predicates that simply reproduce the structure of our SOS and the difficulty is just postponed (it remains to find a stratification for the predicate) or we must have recourse to a very heavy machinery of auxiliary functions (such as in the variant annexed in an earlier draft version of the standard [ISO95]). The result is a much more complex and less intuitive semantics. Moreover, the definition of such predicates would not be the end of our problems. It would remain to prove that this (these) predicate(s) actually meet their definitions, which is not much simpler than our current set of proofs. Finally, the congruence theorem relies on a definition of strong bisimulation which is a priori stronger than the "classical" one [Mil89]: it also includes a condition on the predicates. Hence, one must also prove that this condition is neutral, i.e. that the "stronger" relation is in fact equivalent to the classical one.

Therefore, we propose an ad hoc proof of consistency for our language. Its principle is rather simple: it consists of showing – at the cost of one lemma – that no stratification is needed, i.e. that the RE can be built in just one stratum, which is the most straightforward manner (none of the alternative methods permits to obtain such a simple definition of the RE).

Notations. We use in the sequel the notations of [Gro90b, Gro93], with some

adaptations to our context. A first point we need to clarify is how we instantiate the rules of R. These rules are generic. They are composed of process symbols (P, P', P'', ...) which must be instantiated to get actual transitions. Therefore, σ will denote a substitution of process symbols by actual processes. However, due to data types, we cannot restrict ourselves to substitutions by closed processes only. For example, in $g?x:s;\ P \xrightarrow{g!1} [1/x]P$, process P could be replaced by the open process $b!x;\ stop$, i.e. with the free variable x. We will not enter into tedious syntactic definitions to formalize when substitutions are correct or not. The intuition behind the definition of σ is indeed simple: a correct substitution σ of a given rule replaces all its process symbols by actual processes so that the result is a (static) semantically correct closed behaviour expression.

Let us denote by ρ a rule in R, and by χ a premise or a conclusion in ρ. So, the general format of a rule is:

$$\rho = \frac{\{\chi_k \mid k \in K\}}{\chi}$$

Most rules in R have just one or two premises, but in some cases they can have more. For example in rule (GC3), where $K = \Theta(s)$ and thus depends on the carrier domain of s. The other cases are rules (H3) (K depends on the values of d and Γ) and (En3) (K depends on d). For the axioms, K is void. Note also that for rule (GC3) if $\Theta(s)$ is infinite, and for rules (H3) and (En3) if the time domain is dense, the cardinal of K can be infinite.

Finally, let us denote by ϕ a transition and by σ a "correct" substitution.

Definitions. We define first the intuitive conditions expressed above for RE to agree with R.

RE agrees with R if and only if the following two conditions are met:

Condition 1:

$$\phi \in RE \Rightarrow \left(\exists \frac{\{\chi_k \mid k \in K\}}{\chi} \in R, \exists \sigma \bullet (\sigma(\chi) = \phi \wedge \forall k \in K \bullet RE \models \sigma(\chi_k)) \right)$$

Condition 2:

$$\left(\exists \frac{\{\chi_k \mid k \in K\}}{\chi} \in R, \exists \sigma \bullet (\sigma(\chi) = \phi \wedge \forall k \in K \bullet RE \models \sigma(\chi_k)) \right) \Rightarrow \phi \in RE$$

where:
$RE \models \sigma(\chi_k)$ iff

$$\begin{aligned} &(\quad [\sigma(\chi_k) \text{ is positive} \Rightarrow \sigma(\chi_k) \in RE] \\ &\wedge \quad [\sigma(\chi_k) = P \overset{g}{\not\rightarrow} \Rightarrow (\forall a \in OA \bullet (\underline{\text{name}}(a) = g \Rightarrow \forall P'' \bullet P' \xrightarrow{a} P'' \notin RE))]) \end{aligned}$$

Finally, we define RE. In the following definition, $\aleph_1$ denotes the first cardinal number greater than $\aleph_0 = \omega$ and $RE_{<i}$ is a shorthand notation for: $\bigcup_{0 \leqslant j < i} RE_j$.

Definition 6.1: *RE*

$$RE = \bigcup_{0 \leqslant i < \aleph 1} RE_i.$$

where:

$$RE_i = \left\{ \phi \mid \phi \notin RE_{<i} \right.$$
$$\left. \wedge \left(\exists \frac{\{\chi_k \mid k \in K\}}{\chi} \in R, \exists \sigma \bullet \sigma(\chi) = \phi \wedge (\forall k \in K \bullet RE \models_i \sigma(\chi_k)) \right) \right\}$$

$RE \models_i \sigma(\chi_k)$ iff

$$(\quad [\sigma(\chi_k) \text{ is positive } \Rightarrow \sigma(\chi_k) \in RE_{<i}]$$
$$\wedge \; [\sigma(\chi_k) = P \overset{g}{\not\rightarrow} \Rightarrow (\forall a \in OA \bullet (\underline{\text{name}}(a) = g \Rightarrow \forall P'' \bullet P' \overset{a}{\rightarrow} P'' \notin RE_{<i}))])$$

And we have:

Proposition 6.1: RE agrees with R

We prove below that condition 2 is verified. The proof of condition 1 is more complex and is given in Annex A3. We just outline it below.

Notice first how RE is defined (in just one stage indeed). It can be seen as a pile of layers, a layer (i.e. a RE_i) being a set of transitions. The content of a given layer depends only on the layers below it. Furthermore, due to the first condition in the definition of $RE_i: \phi \notin RE_{<i}$, no two layers contain the same transition. In other words, each transition in RE belongs to one and only one layer. In the sequel we will refer to the index of the layer a transition belongs to (i.e. the i of RE_i) as its level.

So, RE_0 groups all the transitions which can be derived from axioms, RE_1 all the transitions which can be derived from the transitions in RE_0 and so on. This way of doing is natural except for the negative premises. To add a transition to a layer RE_i one simply checks indeed whether the premises are verified on $RE_{<i}$. However, that a negative premise be true on a subset of $RE_{<i}$ does not mean that it remains so on the whole of RE: $P \overset{a}{\rightarrow} P' \notin RE_{<i}$ does not imply $P \overset{a}{\rightarrow} P' \notin RE$. The level of $P \overset{a}{\rightarrow} P'$ might be greater than or equal to i. A priori, condition 1 could thus be violated, i.e. RE could contain a transition whose premises are not actually verified.

Nevertheless, this way of doing is safe in ET-LOTOS. The reason is given by the following lemma (refer to Annex A3 for the proof)

Lemma: $\forall i, j < \aleph_1 \; \forall P, P'' \; \forall \alpha \in A \; \forall d \in D_{0\infty} \; \forall d' < d \bullet$

$$(P \overset{d}{\rightarrow} P_{(d)} \in RE_i \wedge P_{(d')} \overset{\alpha}{\rightarrow} P'' \in RE_j) \Rightarrow i \geqslant j.$$

There is no real intuition behind this lemma. It simply states that the level of a time transition $P \overset{d}{\rightarrow} P_{(d)}$ is greater than or equal to the level of any action transition P may perform within the next d time units (d not included). However, this property is not purely artificial. Basically, it is a consequence of the fact that every "active" process, i.e. which may perform an action, is also subject to the passing of time. What is very logical indeed. Hence the derivation tree of a time transition is always deeper than the derivation tree of an action transition.

Consider now the premises of rule (H3) (the only other rule with negative premises in (En3) and the demonstration would be similar):

$$P \overset{d}{\rightarrow} P_{(d)} \wedge \; \forall g \in \Gamma \; \forall d' < d \bullet P_{(d')} \overset{g}{\not\rightarrow}.$$

A conclusion derived from (H3) may be added to a layer only if $P \xrightarrow{d} P_{(d)}$ is in a lower layer. Hence, as a consequence of the above lemma, all the transitions of the form $P_{(d')} \xrightarrow{\alpha} P'$, with $d' < d$, are also in lower layers.

Proof that Condition 2 is Verified

Let $\frac{\{\chi_k \mid k \in K\}}{\chi} \in R$, and σ a correct substitution such that:

$\sigma(\chi) = \phi \wedge \forall k \in K \bullet RE \models \sigma(\chi_k)$.

By definition of RE, it is enough to prove that: $\exists i < \aleph_1 \bullet (\forall k \in K \bullet RE \models_i \sigma(\chi_k))$.
For then, either $\phi \in RE_i$ or for some $j < i$, $\phi \in RE_j$.
If $\sigma(\chi_k) = P \overset{g}{\not\rightarrow}$, it is obvious that whatever value of $i < \aleph_1$:

$$\forall P' \forall a \in OA \bullet \underline{\text{name}}(a) = g \wedge P \xrightarrow{a} P' \notin RE$$

implies: $\forall P' \forall a \in OA \bullet \underline{\text{name}}(a) = g \Rightarrow P \xrightarrow{a} P' \notin RE_{<i}$

So, it remains to prove that:
$(\forall k \in K \bullet \sigma(\chi_k)$ is positive $\Rightarrow RE \models \sigma(\chi_k))$ implies that for some $i < \aleph_1$:
$(\forall k \in K \bullet \sigma(\chi_k)$ is positive $\Rightarrow RE \models_i \sigma(\chi_k))$
Let $J = \{j \mid \sigma(\chi_k)$ is positive $\wedge \sigma(\chi_k) \in RE_j \wedge k \in K\}$. We can distinguish two cases:

1) $\underline{\max}(J)$ exists. Then, taking $i = \underline{\max}(J) + 1$ suffices.
2) J has no maximum. This is possible if the set of positive premises is infinite. In our TDS, there are 3 rules which can have an infinite number of positive premises (GC3, H3 and En3). We will consider the case of GC3. The proof is similar for the two others:

$$\text{(GC3)} \quad \frac{[tx/x]P \xrightarrow{d+d_1} \forall\, tx \bullet [tx] \in \Theta(s)}{\textbf{Achoice}(d_1)\, x{:}s \,\square\, P \xrightarrow{d} \textbf{Achoice}(d+d_1)\, x{:}s \,\square\, P}$$

Notice that the cardinality of J is at most the cardinality of $\Theta(s)$. If we consider that the abstract data types language used with ET-LOTOS always generates a countable number of data values – which is the case for ACT ONE, then the cardinal of $\Theta(s) \leqslant \omega$.

This is the point where the use of $\aleph_1$ to limit the set of indices in the definition of RE becomes clear. $\aleph_1$ is a *regular* cardinal. This property of regularity implies that any set of ordinal numbers smaller than $\aleph_1$, whose cardinality is also smaller than $\aleph_1$ has a least upper bound smaller than $\aleph_1$. Hence, taking i equal to the least upper bound of J suffices. □

Notice that there are arbitrarily large regular cardinals. Hence, the theory could easily be extended to accept uncountable data types, e.g. $\mathbb{R}^{\geqslant 0} \cup \{\infty\}$ as time domain. The only requirement is that the set of indices in the definition of RE be limited by a regular cardinal greater than the cardinality of any of the data types.

7. Properties of ET-LOTOS

This section is dedicated to a study of the main properties of our model.

7.1. Basic Properties

As the time transitions are used to describe how systems evolve in time, they are expected to possess properties reflecting this intuition. The two propositions below are indeed common to most timing extensions. Their proofs are given in Annex A3.

Proposition 7.1: Time Additivity

$$\forall P, P'' \forall d, d' \in D_{0\infty} \bullet (\exists P' \bullet P \xrightarrow{d} P' \xrightarrow{d'} P'') \Leftrightarrow P \xrightarrow{d+d'} P''.$$

This property is the most intuitive one: if a process may idle for d and then for d' time units, it may idle for $d+d'$ time units and vice versa.

Proposition 7.2: Time Determinism

$$\forall P, P', P'', \forall d \in D_{0\infty} \bullet (P \xrightarrow{d} P' \wedge P \xrightarrow{d} P'') \wedge P' \Rightarrow P''.$$

In ET-LOTOS, as in LOTOS, there is no internal nondeterministic choice operator like e.g. in CSP. Every internal choice is marked by the occurrence of an internal event. Hence an idle process makes no internal choice and then, time transitions are deterministic.

7.2. Strong Bisimulation

Strong bisimulation [Mil 89] is a standard equivalence relation used to compare LTSs. It captures the intuition of a user discriminating processes by observing their behaviours. With strong bisimulation, no difference is made between internal and observable actions.

Basic definitions. Defining the meaning of strong bisimulation in ET-LOTOS is very easy because our underlying model is the usual LTS. Simply, besides being able to observe the actions performed by a process, our user is now also capable to measure the passing of time.

Definition 7.1: Strong Bisimulation

Consider two LTSs $= \langle St_1, A \cup D_{0\infty}, Tr_1, s_{01} \rangle$ and $\langle St_2, A \cup D_{0\infty}, Tr_2, s_{02} \rangle$.

A relation $\mathscr{R} \subseteq St_1 \times St_2$ is a strong bisimulation iff $\forall \langle P, Q \rangle \in \mathscr{R}, \forall v \in A \cup D_{0\infty}$, we have

(i) if $P \xrightarrow{v} P'$, then $\exists Q'$ such that $Q \xrightarrow{v} Q'$ and $\langle P', Q' \rangle \in \mathscr{R}$

(ii) if $Q \xrightarrow{v} Q'$, then $\exists P'$ such that $P \xrightarrow{v} P'$ and $\langle P', Q' \rangle \in \mathscr{R}$

We say that P and Q verify the transfer relation.

This is the classical definition of a strong bisimulation, where time transitions from $D_{0\infty}$ are considered as any other transitions. The strong bisimulation equivalence between two LTS is then defined as follows:

Definition 7.2: $\sim$

Two LTSs $Sys_1 = \langle St_1, A \cup D_{0\infty}, Tr_1, s_{01} \rangle$ and $Sys_2 = \langle St_2, A \cup D_{0\infty}, Tr_2, s_{02} \rangle$ are strong bisimulation equivalent, denoted $Sys_1 \sim Sys_2$, iff

$$\exists \text{ a strong bisimulation relation } \mathscr{R} \subseteq St_1 \times St_2, \text{ such that } \langle s_{01}, s_{02} \rangle \in \mathscr{R}$$

However, only closed ET-LOTOS processes have an LTS and can be compared with each other under the definition above.[8] We now extend this definition to open processes. Let us consider two ET-LOTOS processes P and Q, which contain data variables $\tilde{x}$ at most (we take the convention that a letter topped by a tilde denotes an indexed set). Then, we define $\underline{\text{Sub}}(P, Q)$ as the set of substitutions $[\tilde{t}x/\tilde{x}]$, where $\tilde{t}x$ is an indexed set of ground terms, such that $[\tilde{t}x/\tilde{x}]P$ and $[\tilde{t}x/\tilde{x}]Q$ are closed ET-LOTOS behaviour expressions. So:

Definition 7.3: $\sim_e$

$P \sim_e Q$ iff for any substitution $[\tilde{t}x/\tilde{x}]$ in $\underline{\text{Sub}}(P, Q)$, we have $[\tilde{t}x/\tilde{x}]P \sim [\tilde{t}x/\tilde{x}]Q$.

Notice that $P \sim Q$ implies $P \sim_e Q$. Notice also that $P \sim_e Q$ implies that any free variable which is present in one of the two processes only, has no impact on the LTS of this process. To be more precise: whatever the value given to this variable, all the resulting LTSs are strongly bisimilar. So, if $P \sim_e Q$, we can consider as well that P and Q share the same common set of free variables, all the other variables being instantiated arbitrarily.

A basic requirement about process algebras in general is that strong bisimulation be a congruence. This is very important in order to be able to replace a part of an ET-LOTOS description by another strongly bisimilar process without changing the semantics of the description, i.e. the overall description remains strongly bisimilar to the original one.

As expressed by propositions 7.3 and 7.4 below, this turns out to be the case in ET-LOTOS.

Proposition 7.3:

Let $P \sim_e Q$. Then for any of the following 14 contexts, $C(P) \sim_e C(Q)$:

$C1(\bullet) = \{\Delta^d \bullet\}$ $\quad$ $C2(\bullet) = \{\bullet \Box R\}$
$C3(\bullet) = \{\bullet \,|[\Gamma]|\, R\}$ $\quad$ $C4(\bullet) = \{\textbf{inf} \,|||\, \bullet\}$
$C5(\bullet) = \{\bullet [> R\}$ $\quad$ $C6(\bullet) = \{R [> \bullet\}$
$C7(\bullet) = \{\bullet >> \textbf{accept}\ x_1 : s_1, \ldots x_n : s_n\ \textbf{in}\ R\}$ $\quad$ $C8(\bullet) = \{\textbf{hide}\ \Gamma\ \textbf{in}\ \bullet\}$
$C9(\bullet) = \{[SP] \rightarrow \bullet\}$ $\quad$ $C10(\bullet) = \{go_1 \ldots 0_n @ t[SP]; \bullet\}$
$C11(\bullet) = \{\textbf{i}\{d\}@t; \bullet\}$ $\quad$ $C12(\bullet) = \{\textbf{let}\ x_1 = tx_1, \ldots x_n = tx_n\ \textbf{in}\ \bullet\}$
$C13(\bullet) = \{R >> \textbf{accept}\ x_1 : s_1, \ldots x_n : s_n\ \textbf{in}\ \bullet\}$ $\quad$ $C14(\bullet) = \{\textbf{Achoice}(d)\ x_1 : s_1, \ldots x_n : s_n \Box \bullet\}$

The proof is given in Annex A4.

Proposition 7.4: if $R[g_1, \ldots g_n] \coloneqq P$ and $S[g_1, \ldots g_n] \coloneqq Q$ and $P \sim_e Q$, then $R[h_1, \ldots h_n] \sim_e S[h_1, \ldots h_n]$.

The proof of this proposition is straightforward. It is easy to see that $R[g_1, \ldots g_n] \sim_e P$: the parameters of $R[g_1, \ldots g_n]$ are by definition the free variables of P, and according to (In1, In2) both processes have exactly the same derivatives. Similarly, $S[g_1, \ldots g_n] \sim_e Q$.

Then, $R[g_1, \ldots g_n] \sim_e S[g_1, \ldots g_n]$, what does not depend on the gate names.

[8] So, when we write $P \sim Q$ in the sequel, this means implicitly that P and Q are closed processes.

Furthermore, strong bisimulation is also substitutive under recursion. This is expressed by proposition 7.5 below, whose proof can also be found in Annex A4.

Consider two processes P and Q whose behaviour expressions contain (to simplify) at most one process variable X. Also, P and Q are such that if X is replaced by any process R then: $[R/X]P \sim [R/X]Q$. We denote such a relation between P and Q by $P \sim_p Q$. With such P and Q, it is legitimate to expect that the strong bisimulation be also preserved by recursion, i.e.:

Proposition 7.5: $P \sim_p Q \Rightarrow R \sim S$ where $R := [R/X]P, S := [S/X]Q$.

($[R/X]P$ meaning that the variable X is replaced by the process name R)

The proof is given in Annex A4.

Laws for strong bisimulation equivalence

All the laws listed in section B.2.2 (items a to k) of [ISO 8807] (appendix B) are valid laws for strong bisimulation in ET-LOTOS. The proofs are straightforward.

New equivalence laws can be added to give more insight into the ET-LOTOS operators. The proofs are straightforward.

Urgency		
$\mathbf{i}@t; P$	$\sim \mathbf{i}; [0/t]P$	
$\Delta^d P \Box \mathbf{i}\{d'\}; Q$	$\sim \mathbf{i}\{d'\}; Q$	if $d' < d$
$\Delta^d P [> \mathbf{i}\{d'\}; Q$	$\sim \mathbf{i}\{d'\}; Q$	if $d' < d$
$\alpha\{d\}; P \Box \mathbf{i}\{d'\}; Q$	$\sim \alpha\{d'\}; P \Box \mathbf{i}\{d'\}; Q$	if $d' \leqslant d$
$\alpha\{d\}; P [> \mathbf{i}\{d'\}; Q$	$\sim \alpha\{d'\}; P [> \mathbf{i}\{d'\}; Q$	if $d' \leqslant d$
$\mathbf{exit}\{d\} \Box \mathbf{i}\{d'\}; Q$	$\sim \mathbf{exit}\{d'\} \Box \mathbf{i}\{d'\}; Q$	if $d' \leqslant d$
$\mathbf{i}\{d\}; P \Box \mathbf{i}\{d'\}; P$	$\sim \mathbf{i}\{d'\}; P$	if $d' \leqslant d$
$\mathbf{i}\{d\}; P \Box \Delta^{d'} \mathbf{i}; P$	$\sim \mathbf{i}\{d'\}; P$	if $d' \leqslant d$
Time determinacy		
$\Delta^0 P$	$\sim P$	
$\Delta^d P \Box \Delta^{d+d'} Q$	$\sim \Delta^d (P \Box \Delta^{d'} Q)$	
$\Delta^d P \lvert[\Gamma]\rvert \Delta^{d+d'} Q$	$\sim \Delta^d (P \lvert[\Gamma]\rvert \Delta^{d'} Q)$	
Time additivity		
$\Delta^d \Delta^{d'} P \sim \Delta^{d+d'} P$		
$\Delta^d \mathbf{stop} \sim \mathbf{stop}$		
Persistency		
$a@t; P \Box \Delta^d a@t; [t+d/t]P$	$\sim a@t; P$	
Others		
$a\{d\}; P \Box a\{d'\}; P$	$\sim a\{d\}; P$	if $d' \leqslant d$
$\mathbf{exit}\{d\} \Box \mathbf{exit}\{d'\}$	$\sim \mathbf{exit}\{d\}$	if $d' \leqslant d$
$\mathbf{exit}\{d\} \lvert[\Gamma]\rvert \mathbf{exit}\{d'\}$	$\sim \text{exit}\{d'\}$	if $d' \leqslant d$
$a@t[SP]; P$	$\sim \mathbf{stop}$	if $\nexists\, t' \bullet \vdash [t'/t]\, SP$

7.3. Weak Timed Bisimulation

A second important equivalence relation is the weak bisimulation. It is considered as the basic equivalence in LOTOS: its purpose is to identify processes which seem indistinguishable by an external observation. Thus, contrary to the strong bisimulation, internal actions are not considered as being observable with the weak bisimulation.

Definition 7.4: Weak timed bisimulation
Let $d \in D_{0\infty}$, $a \in OA$ and ε the empty transition:

$$P \stackrel{d}{\Rightarrow} Q \text{ iff } P(\stackrel{\mathbf{i}}{\rightarrow})^* \xrightarrow{d_1} (\stackrel{\mathbf{i}}{\rightarrow})^* \xrightarrow{d_2} (\stackrel{\mathbf{i}}{\rightarrow})^* \ldots \xrightarrow{d_n} (\stackrel{\mathbf{i}}{\rightarrow})^* Q \text{ where } d = \sum_{i=1}^{n} d_i$$

$$P \stackrel{a}{\Rightarrow} Q \text{ iff } P(\stackrel{\mathbf{i}}{\rightarrow})^* \stackrel{a}{\rightarrow} (\stackrel{\mathbf{i}}{\rightarrow})^* Q$$

$$P \stackrel{\varepsilon}{\Rightarrow} Q \text{ iff } P(\stackrel{\mathbf{i}}{\rightarrow})^* Q$$

Consider two LTSs $= \langle St_1, A \cup D_{0\infty}, Tr_1, s_{01} \rangle$ and $\langle St_2, A \cup D_{0\infty}, Tr_2, s_{02} \rangle$.

A relation $\mathscr{R} \subseteq St_1 \times St_2$ is a weak timed bisimulation iff $\forall \langle P, Q \rangle \in \mathscr{R}$, $\forall v \in OA \cup D_{0\infty} \cup \{\varepsilon\}$:

(i) if $P \stackrel{v}{\Rightarrow} P'$, then $\exists Q'$ such that $Q \stackrel{v}{\Rightarrow} Q'$ and $\langle P', Q' \rangle \in \mathscr{R}$

(ii) if $Q \stackrel{v}{\Rightarrow} Q'$, then $\exists P'$ such that $P \stackrel{v}{\Rightarrow} P'$ and $\langle P', Q' \rangle \in \mathscr{R}$

Definition 7.5: $\approx$
Two LTSs $Sys_1 = \langle St_1, A \cup D_{0\infty}, Tr_1, s_{01} \rangle$ and $Sys_2 = \langle St_2, A \cup D_{0\infty}, Tr_2, s_{02} \rangle$ are strong bisimulation equivalent, denoted $Sys_1 \approx Sys_2$, iff

$\exists$ a weak timed bisimulation relation $\mathscr{R} \subseteq St_1 \times St_2$, such that $\langle s_{01}, s_{02} \rangle \in \mathscr{R}$

Proposition 7.6:

$P \sim Q$ implies P $\approx Q$

The proof is straightforward.

Equivalence laws

$P \approx \mathbf{i}; P$ (Remember that $\mathbf{i}; P$ is a shorthand notation for $\mathbf{i}\{0\}; P$)
$P \Box \mathbf{i}; P \approx \mathbf{i}; P$
$a; (P \Box \mathbf{i}; Q) \Box a; Q \approx a; (P \Box \mathbf{i}; Q)$
$\mathbf{i}\{d\}; \mathbf{i}\{d'\}; P \approx \mathbf{i}\{d + d'\}; P$

Note that $\approx$ is not a congruence in front of $\Box$ and $[>$, like the weak bisimulation in LOTOS, but also in hiding[9] and enabling contexts. For example:

$\mathbf{i}\{d\}; a; b\{0\}$ **stop** $\approx a; b\{0\};$ **stop**
but **hide** a **in** $(\mathbf{i}\{d\}; a; b\{0\};$ **stop**$) \not\approx$ **hide** a **in** $(a; b\{0\};$ **stop**$)$.

In LOTOS, it is easy to find the weakest congruence stronger than $\approx$, which is not the case in ET-LOTOS. The weakest congruence stronger than $\approx$ is still to be found.

7.4. Conservative Extension

Finally, we address the problem of the compatibility between the theories of LOTOS and ET-LOTOS. A desirable property of a timing extension of an existing formalism is that it should depart as little as possible from the (untimed) language. Two main motivations justify this requisite. The first one is the comfort of the

[9] As pointed out in [CdO94].

specifier which should move easily from an untimed to a time environment. As far as possible, the properties and the intuitive understanding in the untimed context should be preserved in the timed one. It should be possible for instance, to refine an untimed specification into a timed one. The second reason is theoretical. It is interesting to benefit in the timed model from the theoretical results already obtained for (untimed) LOTOS. Both semantics should thus be as similar as possible.

These two requirements are generally encompassed in the notion of "conservative extension" [GrV 92]. We translate here to the LOTOS framework the definition of this property given in [NiS 92].

Consider the LOTOS process algebra LOTOS $= (OP, A, R_A^{OP}, \sim)$ where OP is a set of operators, A is the alphabet of actions, R_A^{OP} is the set of operational semantics rules and $\sim$ the strong bisimulation equivalence. Consider ET-LOTOS as the process algebra ET-LOTOS $= (OP', A', R_{A'}^{OP'}, \sim_E)$ where OP' is a superset of OP, $A' = A \cup D$ is a superset of A, $R_{A'}^{OP'}$ is the new set of rules, and $\sim_E$ is the strong bisimulation equivalence in ET-LOTOS ($\sim_E$ denotes our $\sim$ as defined in Section 7.2).

Then, ET-LOTOS is a conservative extension of LOTOS if the next two requirements are met:

- Semantics conservation (also called operational conservative extension): $\forall \rho \in R_A^{OP}$, ρ is valid in $R_{A'}^{OP'}$ if it is applied on **LOTOS** terms.

  The rules R_A^{OP} remain valid in ET-LOTOS as far as they are applied on LOTOS terms.

- Isomorphism (also called operational conservative extension up to bisimilarity): $\forall P, Q \in \mathbf{LOTOS} \bullet P \sim Q \text{ iff } P \sim_E Q$.

  The theory of processes in LOTOS is isomorphic to that of the restriction of ET-LOTOS to constructs of LOTOS.

The semantics conservation is easy to check, keeping in mind that the ET-LOTOS extensions are optional ($@t$) or that the LOTOS operators are shorthand notations (**i**; P for **i**$\{0\}$; P, **exit** for **exit**(∞)).

As to the isomorphism of the (ET-LOTOS, $\sim_E$) and the (LOTOS, $\sim$) theories, we prove in Annex A5 that it is true for guarded[10] specifications.

With unguarded specifications however the isomorphism between the theories is not true anymore. For example, in LOTOS, $P :=$ **stop** and $Q := Q$ are strongly bisimilar, whereas in ET-LOTOS, $P \xrightarrow{d}$ but $Q \not\xrightarrow{d}$. Note that discriminating these two processes is considered more as an asset than as a shortcoming. Furthermore, this distinction is due to the fact that unguarded recursions block time. This is indeed common to almost all the timed formalisms.

7.5. Unrealistic Behaviours

ET-LOTOS permits the description of behaviours that no existing system can exhibit. We did not express restrictive conditions on the use of ET-LOTOS to avoid such behaviours for two main reasons. Firstly, these unrealistic behaviours did not introduce extra complexity in the semantics. Our model associates an LTS with every syntactically correct process and there is no need to perform some prior

[10] The notion of guarded specification is defined in Annex A5.

verifications. Considering the subtleties associated with these processes, it would have been much more complicated to bar them than to cope with them. Secondly, some of these behaviours are useful abstractions and the ability to describe them is considered as an asset. Note finally that, though "unrealistic", the behaviours of the processes which we describe in sections 7.5.1 and 7.5.2 below are nevertheless coherent with their definitions. In other words, processes do not behave strangely if they are not specified to do so.

7.5.1. Infinite Variability

The property of finite variability states that a process can perform only finitely many actions in a finite time. It is indeed a logical requirement to expect from the description of any realistic system. In most timed process algebras however, processes can be specified which do not verify that property.

In ET-LOTOS for example, let $P := a; P$. P may perform an infinite number of actions at a same time: $P \xrightarrow{a} \xrightarrow{a} \ldots$. With a dense time domain, P may also perform an infinite number of actions in a finite time: $P \xrightarrow{a} \xrightarrow{1/2} \xrightarrow{a} \xrightarrow{1/4} \xrightarrow{a} \xrightarrow{1/8} \ldots$.

The formalisms preventing such behaviours either associate a delay with the recursive calls [ReR 88] or insist that the recursive calls be guarded by some delay in the recursively defined processes.

However, the ability to specify processes that do not respect the finite variability property can be considered as an asset. Even though they have no practical use when considered in isolation, they can be useful when integrated into larger processes. This is the case in particular in the constraint-oriented approach [Bri 89], where it is required that the different constraints on a process behaviour be clearly separated and described by different processes. Then, some constraints can simply be on the logical ordering of the events without including any unwanted timing information. For example $P := a; b; P$ just specifies the property that a and b are performed alternately (starting with a).

We can distinguish between the infinite behaviours that can potentially occur in zero time and in a finite but non zero time. Therefore, we introduce here a classification, which is inspired by the one given in [DBS96].

A *spin behaviour* is an infinite sequence of actions at a same time. As regards the infinite sequences of actions occurring in a finite non zero time, they are generally called "zeno" (with reference to the well-known paradox of Zeno of Elea including Achille and the tortoise). Hence we could simply define a *zeno behaviour* as an infinite sequence of actions in a finite non zero time. Similarly, a *spin divergence* is an infinite sequence of internal actions at the same time, and a *zeno divergence* as an infinite sequence of internal actions in a finite non zero time.

However, we will extend a bit these definitions to include another kind of "unrealistic" behaviour. Consider the following process:

$$Q := Q(1) \textbf{ where } Q(x) := \Delta^{x/2}; Q(x/2).$$

Q has a time transition for every time less than *1*, but not for *1*. In other words, the set of possible time transitions of Q is right-open. Hence, Q may idle but without ever being able to reach time *1*. We will call *pure zeno* the behaviour of a process which (courageously) idles towards a finite time it cannot reach, e.g.: $Q \xrightarrow{1/2} \xrightarrow{1/4} \xrightarrow{1/8} \ldots$

To describe a process exhibiting such a behaviour, we introduce a shorthand notation: **zeno**(d), with $d \in D_{0\infty}$, is a process which can only perform time transitions of a duration less than d. The definition of **zeno**(d) is simply $Q(d)$ above.

From the viewpoint of an external observer, $\mathbf{zeno}(1) \xrightarrow{1/2} \xrightarrow{1/4} \xrightarrow{1/8} \ldots$ cannot be distinguished from e.g. the zeno divergence $P \xrightarrow{\mathbf{i}} \xrightarrow{1/2} \xrightarrow{\mathbf{i}} \xrightarrow{1/4} \xrightarrow{\mathbf{i}} \xrightarrow{1/8} \ldots$ Hence we also integrate the pure zeno behaviours into the zeno divergences.

Notice finally that the choice of the time domain is not neutral in the appearance of pure zeno behaviours. Consider for example the following process:

$$P := \mathbf{hide}\ g\ \mathbf{in}\ g@t\,[t^2 \geqslant 2]; Q$$

In case the time domain is $D = \mathbb{R}^{\geqslant 0} \cup \{\infty\}$, P can idle for $2^{1/2}$ time units and then, perform the hidden g. However, P is $\mathbf{zeno}(2^{1/2})$ if the time domain is $D = Q^{\geqslant 0} \cup \{\infty\}$: P cannot perform a transition of duration $2^{1/2}$, as this value does not belong to D, and any time transition greater than $2^{1/2}$ is forbidden as well because it would pass beyond the occurrence time of an urgent g.

7.5.2. Timelocks

Some ET-LOTOS processes may not idle. We have already seen useful examples of this capability: it allows one to ensure the occurrence of internal events within a finite time. For example: **i**; **stop** may not idle. However, this process may freely perform **i** and thereby be able to idle anew. As a matter of fact, ET-LOTOS also allows the specification of processes which may not continue to idle "by themselves", i.e. simply by performing internal events. We will distinguish, then, between two definitions of time blockages. What we will call a *time blockage* from now on is simply a state where the process may not idle. For example: **i**; P is in a time blockage.

We will call *timestop* the situation of a process which may neither age nor perform any internal transition. Finally a timestop process that cannot perform any observable action is in a *timelock* state. For example, **block** is a timelock.

We detail in the sequel different situations leading to timelocks.

Initially Unguarded Specifications. A first category of processes causing timelocks is the one of initially unguarded specifications.[11] One cannot derive any time transition for such specifications: this leads to an infinite computation.

This can be explained easily by the definition of RE. Consider the simple case P **where** $P := P$. Imagine that there is one i such that $P \xrightarrow{d} P_{(d)} \in RE_i$. Hence, by definition of RE_i, there is a rule in R from which $P \xrightarrow{d} P_{(d)}$ is derived. The only possible rule is (In2). Hence, there is a $j < i$ such that the premise of (In2), i.e. $P \xrightarrow{d} P_{(d)}$, belongs to RE_j. And so on.

Eventually one finds $P \xrightarrow{d} P_{(d)} \in RE_0$, which is not true.

That initially unguarded specifications block time should not be a severe problem, such specifications being not very useful usually (e.g. P **where** $P := Q \Box P$).

However, there is an exception: Ps **where** $Ps := P \,|||\, Ps$. Such a construct is often used in LOTOS specifications of services and protocols for instance, when one does not want to restrict a priori the number of possible instances of P running in parallel. This is the reason why we have introduced the **inf** $|||$ operator. Unlike Ps **where** $Ps := P \,|||\, Ps$, the specification $Ps' := \mathbf{inf} \,|||\, P$ does not block time (provided P does not).

Notice finally the case of weakly guarded (i.e. guarded by an internal event) processes, which lead to divergence. For example, $P := \mathbf{i}; P$. This process may never idle: it always remains in a time blockage as it must always perform urgent **i** actions.

[11] The notions of guarded and initially guarded specifications are formally defined in Annex A5.

Timelocks with a Dense Time. Some timelocks can only appear *when time is dense*.[12] We give below examples of these cases. As we will see, the mechanism is always basically the same.

*With **hide***

hide enforces the immediate occurrence of any hidden action. A problem arises if the first time a hidden action is enabled is undefined. A non cautious use of selection predicates makes this possible with a dense time domain.

Example 1:

$$\textbf{hide}\ g\ \textbf{in}\ (g@t[t > 1]; \textbf{stop})$$

the predicate $[t > 1]$ enables action g in the left-open time interval $(1, \infty)$: action g is enabled at any time after *1*, but not at time *1*. After having idled for one time unit, the process faces a timelock: neither can it idle any further, nor can it release the timelock by accomplishing the hidden action g, g being not enabled yet at time *1*.

Example 2:

$$\textbf{hide}\ g\ \textbf{in}\ (\textbf{choice}\ n : Nat \Box g@t[t = (1/2)^n]; \textbf{stop})$$

It can be seen easily that this process also faces a timelock. This example is just a more subtle version of the previous one. The hidden process does not offer g at time *0*. Neither does it offer g continuously after time *0*. However, any time transition d would pass beyond the occurrence time of a hidden g.

Example 3:

$$\textbf{hide}\ g\ \textbf{in}\ (g?x : time?y : time@t[x = t+4 \wedge y = 2{*}t \wedge < y]; \textbf{stop})$$

This process faces a timelock after *4* time units, for the same reasons as the previous ones. This example shows how a timelock can arise from the interaction in a predicate between constraints on t and constraints on other attributes.

Example 4:

$$\textbf{hide}\ g\ \textbf{in}\ (g?x : time@t[(x = 0 \wedge t = 0) \vee (x = 1 \wedge t > 0)]; \textbf{stop}\ |[g]|\ g!1; \textbf{stop})$$

This example insists on the attention which must be paid to the difference between the notions of gates and of actions. The process faces a timelock immediately, although the predicate always allows actions on gate g. However, action $g!1$ is only enabled in the interval $(0, \infty)$.

*With **choice over values***

A timelock similar to the ones presented above appears with a process like:

$$\textbf{choice}\ x : time \Box [x > 0] \rightarrow \Delta^x \mathbf{i}; \textbf{stop}$$

This is because there is no earliest internal action: whatever the delay x is, there always exists a smaller delay (another x) that enables an urgent **i** at that time.

Conclusion on timelocks with a dense time. All the previous processes face a timelock, due to the density of time, because the first time at which an urgent event is enabled is undefined. Such situations are caused by a non careful use of gates or

[12] This has been noticed, and signalled to us, by Steve Schneider.

selection predicates. In particular, danger arises when these predicates enable actions on left-open time intervals.

8. Related Work

8.1. Comparison with Basic ET-LOTOS

[LéL94][13] presented a first version of ET-LOTOS, restricted to the basic language (i.e. without data types). It already included the life reducer and $@t$ but of course no selection predicate. From the semantic point of view, the upgrade to the current version has not been a trivial task. One of the difficulties was the provision of the choice over values with an adequate semantics. Our solution meets the intuition of the operator and requires no restriction on its use but uses infinite premises in some cases. A more constraining proposal is discussed in Section 8.3.

Another major evolution was the withdrawal of the reverse persistency property. It is a typical example of the dilemmas we faced when designing ET-LOTOS between our desire of an expressive formalism and our will to keep the semantics simple.

The reverse persistency property states that no new action is enabled while a process idles. Formally: $P \xrightarrow{d} P' \wedge P \overset{g}{\not\rightarrow} \Rightarrow \forall\, d' < d \bullet P_{(d')} \overset{g}{\not\rightarrow}$.

This property allowed an appreciable simplification of the rules for **hide** and the enabling. Remember the premises of rule (H3):

$$P \xrightarrow{d} P', \forall g \in \Gamma \bullet P \overset{g}{\not\rightarrow}, (\forall g \in \Gamma\, \forall d'' < d \bullet P \xrightarrow{d''} P'', P'' \overset{g}{\not\rightarrow}).$$

hide Γ **in** P must not idle beyond the time a hidden action can occur. With the reverse persistency, it is indeed sufficient to verify that no hidden action is enabled initially. Hence, the premise of rule (H3) in [LéL94] was simply: $P \xrightarrow{d} P', (\forall g \in \Gamma \bullet P \overset{g}{\not\rightarrow})$.

However, a first problem with reverse persistency (already in basic ET-LOTOS) is that it is in general not compatible with time additivity.

For example, we may have: $\Delta^5 a;\mathbf{stop} \xrightarrow{5} \Delta^0 a;\mathbf{stop} \xrightarrow{1} a;\mathbf{stop}$,
but not: $\Delta^5 a;\mathbf{stop} \xrightarrow{6} a;\mathbf{stop}$.

Furthermore, in (full) ET-LOTOS we had to choose between this property and the expressive capabilities offered by the use of $@t$ with selection predicates. It is clear for example that a process like: $g?x{:}s@t\ [x = t];P$ violates the property (remember that an action is determined by a gate name and by the value of its attributes: $g!1, g!2, g!0.5$ are different actions): reverse persistency is not compatible with a continuous creation of actions.

Hence, our decision has been to preserve the expressive power offered by $@t$, which is mandatory for the description of some common mechanism as we saw in Section 5. This also allowed us to get back time additivity.

Another argument supporting this choice is that it does not actually bring any additional level of complexity in the semantics. Basically, in rule (H3), $P \xrightarrow{d''} P'' \Rightarrow P'' \overset{g}{\not\rightarrow}$ can be seen as a boolean function which must be verified on the domain $[0, d)$. However, computations of the same complexity must still be performed in the presence of the reverse persistency.

[13] A comparison with earlier proposals, like [LeL93], can be found in that paper.

For example, **hide** g **in** $g@t[SP(t)]; P \xrightarrow{d}$ requires to verify that $SP(t)$ is never true on $[0, d)$. It is not more complex than verifying that **hide** g **in** $g?x{:}s[SP'(x)]; P \xrightarrow{d}$, i.e. that $SP'(x)$ is never true on $\Theta(s)$. In fact, such a level of complexity cannot be avoided if one wants a way to express urgency.

8.2. LOTOS-T and its Successors

LOTOS-T [MFV 94] proposes a timed version of full LOTOS based on a minimal extension of LOTOS. Neither does it offer a delay operator, like Δ, nor a specific time measurement operator, like $@t$. Time is solely introduced in the specifications by way of time labels added to the actions: for instance, $a(5); P$. These labels have a more restrictive meaning than the life reducer: in LOTOS-T, $a(d); P$ means that a is only enabled at time d precisely, neither before, nor after.

Despite its simplicity, LOTOS-T offers expressive capabilities comparable to the ones of ET-LOTOS: it is possible to measure time duration and to link the attributes of an action with the passing of time. Furthermore, the language has the maximal progress property. But the specifications are often more complex. In particular, one has to resort extensively to the **choice over values** operator.

For example: $a@t[SP(t)]; P(t)$ is written in LOTOS-T as: **choice** $t{:}time\,\square[SP(t)] \rightarrow a(t); P(t)$.

Similarly, a nondeterministic delay is described by: **choice** $t{:}time\,\square[min \leqslant t \leqslant max] \rightarrow \mathbf{i}; \mathbf{i}(t); P$. The first **i** is needed to resolve the nondeterministic choice, otherwise $\mathbf{i}\{min\}$ would always be selected. This is sometimes a drawback. Consider for instance the following ET-LOTOS specification of an insecure system which can crash at any time: $System\ [> \mathbf{i}\{\infty\}; \mathbf{stop}$. This is because the life-reducer $\{\infty\}$ introduces a nondeterministic delay in $[0, \infty)$, before the occurrence of the internal action that disables *System*. This specification works precisely because no **i** is required to occur immediately.

The main difference with ET-LOTOS lies in fact in the semantics. Each process is also associated with an LTS but the transition derivation system is not presented in two columns with a clear separation of concerns between the progression in time and the execution of actions. Both are intertwined because the authors use an alphabet of extended actions composed of a usual action and of a time stamp that indicates the (relative) time at which the action can be performed. For example, $a(5); P$ can do just one transition: $a(5); P \xrightarrow{a5} P$. Hence, there are no transitions dedicated to the description of the passing of time. For example, **stop**, **block** and Δ^d **block** cannot be distinguished: these three processes have the same LTS with just one state and no transition as they can perform no action. In fact there is no process such as **block** in this approach: it is implicitly assumed that a process is always capable to idle. In particular, unguarded recursions should not block the passing of time. The only needed restriction is to enforce urgency on internal actions.

The difficulty with this model comes from the mechanism used to ensure this urgency. In [MFV94] the semantics is given in two steps. In the first step, an unrestricted transition system is generated in a standard manner without considering urgency. In the second step, a restricted transition system is derived by removing transitions that do not fulfil the urgency condition: a transition labelled at (with a being observable or internal) is removed if a transition $\mathbf{i}t'$ with $t' < t$ originates at the same state. However, this definition lacks compositionality in some cases, in the sense that the *restricted* transition system of a composed behaviour

cannot be derived from the *restricted* transition systems of its components (but from the *unrestricted* ones).

Consider for example the process: $P(1)$ **where** $P(t) \coloneqq \mathbf{i}(t); Q \,|||\, P(t/2)$.[14]

After the first step there is one transition $\mathbf{i}t$ leaving $P(1)$ for every t of the form $(1/2)^n$. After the second step, there remains no transition at all: for every $\mathbf{i}t$ there is an $\mathbf{i}t'$ with $t' < t$. Hence, $P(1)$ has the same LTS as **stop**.

However, $a(1)$; **stop** $|||$ $P(1)$ has not the same LTS as $a(1)$; **stop** $|||$ **stop**. The latter can perform $a1$ but not the former: transition $a1$ is withdrawn from the LTS at the second step because of the $\mathbf{i}t$ transitions of $P(1)$. So, strong bisimulation is not a congruence in this model.

[QMF94] proposes a new version of the formalism inspired by ET-LOTOS. It also contains (with a different syntax) a delay operator, a life reducer (on observable actions only) and an $@t$ construct, but no selection predicate on action-prefix. Its semantics is still based on an alphabet of extended actions, as in [MFV94], but is now defined in one step. Therefore, negatives premises are included in the TDS to ensure the urgency of internal actions, e.g.

$$\frac{P \xrightarrow{at} P', \forall\, t' < t \bullet Q \overset{\mathbf{i}t'}{\not\rightarrow}}{P \,|[\Gamma]|\, Q \xrightarrow{at} P' \,|[\Gamma]|\, \mathbf{Age}(Q, t)} \quad (\underline{\mathrm{name}}(a) \notin \Gamma \cup \{\delta\})$$

Notice also the use of an auxiliary **Age** operator. However, this semantics is still in trouble with processes like the above example. It is not clear indeed what the behaviour of $P(t)$ should be. The solution "$P(t)$ cannot perform any transition" is not valid: either $P(t)$ can perform $\mathbf{i}t$ because $P(t/2)$ cannot perform any $\mathbf{i}t'$ with $t' < t$, or $P(t/2)$ can perform an $\mathbf{i}t'$ with $t' < t$ and then also $P(t)$. The first option does not seem logical: if $P(t)$ can perform $\mathbf{i}t$, by symmetry, $P(t/2)$ should be able to perform $\mathbf{i}t/2$. Hence, $P(t/2)$ can perform $\mathbf{i}t'$ with $t' < t$. But what could be the value of t'? Either $t' = t/2$ and $P(t/4)$ cannot perform any $\mathbf{i}t''$ with $t'' < t'$ which is not logical, or $P(t/4)$ can perform $\mathbf{i}t'$ with $t' < t/2$, and then $P(t/8)\ldots$.

Finally, to solve these problems and thereby align LOTOS-T with ET-LOTOS, a revised but more complex version has been proposed. It is still based on timed-actions but in order to distinguish between **stop**, **block**, Δ^d **block**, this semantics is completed with an auxiliary boolean functions $\underline{\mathrm{Ci}}(d, P)$, associated with each state of the LTS and which indicates whether the process in state P can idle for d time units or not. $\underline{\mathrm{Ci}}(d, P)$ is in fact borrowed from an earlier version of ET-LOTOS.

8.3. Timed Full LOTOS

A timed extension of Basic LOTOS, Timed Basic LOTOS, is proposed in [BLT94a]. In [BLT94b], it is extended to full LOTOS and called Timed Full LOTOS. The comparison between Timed Full LOTOS and ET-LOTOS is interesting because these authors faced the same problems as we did but they opted in general for different solutions. Basically they put more stress on keeping the semantics simple. A second important objective was also, taking up the terms of [BLT94a], to define a "timed-interaction" extension of LOTOS, whereas ET-LOTOS is a "timed-action" extension (we explain these terms in the sequel). Another important difference with ET-LOTOS is the policy with respect to urgency. Timed Full

[14] $\mathbf{i}(t)$ means that $\mathbf{i}$ is enabled and urgent at time t only.

LOTOS is not based on the maximal progress property, but on operators that can make actions urgent. Finally, some restrictions are imposed on the use of the data values in Timed Full LOTOS.

We detail below the main characteristics of Timed Full LOTOS and compare them with the ones of ET-LOTOS.

Timed-action vs. timed-interaction

In Timed Full LOTOS, two operators are introduced to express timed features: **time** $g(d_1, d_2)$ **in** P and **time-once** $g\,(d_1, d_2)$ **in** P. The meaning of these operators is simple to understand: they restrict to the interval (d_1, d_2) the period when any action on gate g is enabled by P. The first operator does it repeatedly, the second one just for the first occurrence of g in P. The time values d_1 and d_2 refer to the time at which g is enabled in P.

For example, in: **time** $g(2,4)$ **in** $(b; g; P\,|[g]|\,g; Q)$, g is enabled between *2* and *4* time units after the occurrence of b. These operators follow a must-timing policy: if g is still enabled at the expiration of the time interval, time is blocked.

Notice the difference with ET-LOTOS: **time** $g(d_1, d_2)$ **in** P and **time-once** $g(d_1, d_2)$ **in** P are new operators, and not just enhancements of the action prefix like $@t$ or the life reducer. Such operators are potentially more powerful. They can be applied to a single action, for example: **time-once** $g\,(d_1, d_2)$ **in** g; P, but they can be applied to the result of an interaction as well. Try for example to express that g should be enabled for just one time unit in the process: b; g; **stop** $|[g]|\,c$; g; **stop**. With Timed Full LOTOS, this is specified with the operator **time** $g\,(0, 1)$ **in** .... With ET-LOTOS, this cannot be done simply with an operator or by adding a new constraint in parallel. A solution is:

> **hide** *sync* **in** $(b; sync; g;$ **stop** $|[g, sync]|\,c; sync; g;$ **stop** $|[g, sync]|\,sync; g\{0, 1\};$ **stop**).

In addition to the new constraint on g, an internal synchronisation is required to determine the enabling time of g. Note however that the Timed Full LOTOS and the ET-LOTOS solutions are not equivalent, as Timed Full LOTOS follows a must-timing policy.

One could imagine providing ET-LOTOS with similar operators, for example by extending **hide** with a delay parameter, e.g. **hide** $g\,(d_1, d_2)$ **in** P, meaning that g must occur between d_1 and d_2 time units after its enabling by P. In [LéL94] we have already discussed this possibility. The problem with such operators is that their semantics is very complex and rather tricky, unless the formalism possesses the reverse persistency and also the persistency property. This last property states that: $P \xrightarrow{d} P'$ and $P \xrightarrow{a}$ imply $P' \xrightarrow{a}$. Intuitively, no action is disabled when a process idles. Like the reverse persistency, keeping the persistency property is demanding. For example, transitions like $a\{2\}; P \xrightarrow{3}$ **stop** would not be allowed. And a continuous disabling of actions is not possible either. For example, $g?x{:}s\,@t\,[x = t]; P$ clearly violates the property.

Hence, we decided not to include such operators in ET-LOTOS. Another reason supporting this choice is that we lacked practical examples where this would bring a real advantage.

As a matter of fact however, Timed Full LOTOS has both properties of persistency and reverse persistency. This implies the loss of time additivity and forbids the continuous creation or disabling of actions. In fact, no mechanism similar to $@t$ is defined in Timed Full LOTOS.

Urgency

Timed Full LOTOS only proposes timing operators following a must-timing policy. Even though urgent observable actions are considered mandatory, operators with a may-timing policy (i.e. which do not block the passing of time) would be useful too in our opinion.

Urgent observable events are indeed presented in [BLT94b] as necessary to describe "special" actions, whose occurrence should be independent of the environment. For "common" observable events, however, urgency is meaningless. But the blockage of time introduced by the two operators of Timed Full LOTOS remains. In [BLT94a], the authors suggest a kind of time-out mechanism to avoid this undesired blockage and simulate may-timing.

For example: $a(d_1, d_2); P \Box \mathbf{i}(d_2, d_2); \mathbf{stop}$.[15]

Yet, such a solution is not completely adequate. Introducing an **i** is not convenient because it resolves choices. For example, we wonder how a simple ET-LOTOS process like $P[> a\{1\}; Q$ could be described in Timed Full LOTOS.

Furthermore, it is not possible to specify correctly punctual non urgent actions. Writing $a; Q \Box \mathbf{i}(0); \mathbf{stop}$ is not satisfactory in this approach: even though the environment offers a the interaction can be prevented by the occurrence of **i**. Notice also that this time-out mechanism can only be applied with action prefixes, not with interactions.

Note finally that the must-timing policy is indeed necessary in Timed Full LOTOS: the persistency property holds because time is blocked at the expiration of the delay specified by the timing operators.

The choice over values

The idea of the authors in [BLT94b] is to avoid the possibly infinite premise of our rule (CH2). So they require that, in every construct like **choice** $x:s \Box P(x)$, $P(x)$ be able to age independently of the value of x. Then, the rule is simply:

$$\frac{P(x) \xrightarrow{d} P'(x)}{\mathbf{choice}\, x:s \Box P(x) \xrightarrow{d} \mathbf{choice}\, x:s \Box P'(x)}$$

where the value of x in $P(x) \xrightarrow{d} P'(x)$ does not matter.

It is not clear however how $P'(x)$ is determined, unless the axioms and inference rules of the semantics also accept open behaviour expressions. Furthermore, defining a sufficient static semantic restriction on $P(x)$ is not easy. The criterion given in [BLT 94b] is not sufficient in our opinion. It says that "any sub-expression '**time** $a(E_1, E_2) \ldots$' appearing in P, where E_1 and/or E_2 contain an occurrence of variable x, which is free in P, must be guarded in P by an action prefix or by an enabling operator". But, for example, let $P(x) :=$ **time** $a(d_1, d_2)$ **in** $a?y:s[SP(y, x)]; Q$. $P(x)$ meets the requirement and normally, time should be blocked after d_2 time units. However, for values of x such that no y makes SP true, time is not blocked after this delay. The way $P(x)$ ages depends thus on the value of x which may then not be chosen arbitrarily. A sufficient criterion might be to extend the restriction on **time** $a(E_1, E_2)$ to the gates and the selection predicates.

Restrictions on the use of time values

As explained above, Timed Full LOTOS has the persistency and reverse

[15] $a(d_1, d_2); P$ is a shorthand notation for **time-once** $a(d_1, d_2)$ **in** $a; P$. In the sequel, we also use the notation $a(d); P$ for **time-once** $a(d, d)$ **in** $a; P$.

persistency properties, which forbid a continuous creation or disabling of actions. So, no mechanism like $@t$ combined with the selection predicate is proposed. The choice over values operator, used in the same way as in LOTOS-T, could provide another way to violate these properties.

For example: **choice** $x{:}time \Box a!x(x, \infty); P$ violates the reverse persistency property.

However, we saw above that Timed Full LOTOS expresses restrictions on the use of the choice over values that precisely preclude such specifications. Then, there is no simple way in Timed Full LOTOS to measure time duration. One has to resort to more complex constructs.

8.4. RT-LOTOS

RT-LOTOS [CdO94] is extension of ET-LOTOS with two new operators aimed at improving two aspects of our formalism.

Firstly, Ω^d is aimed at expressing nondeterministic delays without introducing an internal event. However, it turns out that Ω^t can do so only in the case: $\mathbf{i}\{d\}; a; P$, i.e. when the nondeterministic delay directly prefixes another action prefix. For example, no replacement would be possible in the case $\mathbf{i}\{d\}; (a; P \Box b; Q)$. Furthermore, this second action prefix must be time-independent, i.e. it cannot carry a time-measurement variable, nor a life reducer.

In the other cases, it is possible to replace $\mathbf{i}\{d\}; P$ by a similar mechanism expressed with Ω^d, namely: $(\Omega^d \textbf{\textit{exit}}) >> P$. Yet, this brings no benefit: this construct also introduces an **i** and writing it is less user friendly than writing $\mathbf{i}\{d\}$. On the other hand, the definition of Ω^d requires the decoration of all the events in the semantics with an additional subscript.

Secondly, the so-called "temporal disrupt operator", denoted $a\{d, Q\}; P$, starts an exception behaviour Q when a time constraint $\{d\}$ on some external action is not matched by the environment. $a\{d, Q\}; P$ behaves like $a\{d\}; P \Box \Delta^d Q$, except at time d: in this case $a\{0, Q\}; P$ ensures that the occurrence of a still has the priority on any action of Q; which is not the case with $a\{0\}; P \Box \Delta^0 Q$. So, $a\{0, Q\}; P$ allows them to express that Q is enabled *only if* the environment is not ready to interact on a. This effect is obtained at the expense of adding in the semantics a new special event π of lower priority than the other actions and with some new rules dedicated to its treatment. In fact, this temporal disrupt operator introduces some form of priorities in LOTOS, but in a very restricted context. In our opinion, it would be better to deal with priorities at a more general level. This is the reason why we consider this issue as being beyond the scope of ET-LOTOS.

8.5. Other Timed Process Algebras

Besides the extensions of LOTOS which we have just discussed, many other timed process algebras have been proposed in the recent years. In this section, we will consider extensions of CCS, of CSP and of ACP and a last extension of (basic) LOTOS. In some of them, the timing constructs may be parametrized with time variables, but other data types are not considered in the semantics.

The next two formalisms share many common points with ET-LOTOS. Both are based on the assumption of a dense time (the non-negative reals) and have the maximal progress property. Both also propose a mechanism to measure the passing of time.

Timed CCS [Wan91]

We have borrowed $@t$ from this language, where it was originally introduced to define an expansion theorem.

Timed CCS has an independent delay operator, as in ET-LOTOS. A difference with our language is that it has the persistency property but no operator like the life reducer. To describe an action proposed for a limited time, a time-out like mechanism with an urgent **i** is thus necessary, e.g. $a; P \Box \Delta^d \mathbf{i}; \mathbf{stop}$. Resorting to this mechanism could be cumbersome in some cases, for instance to describe a process like $a\{3\}; P' \Box b\{4\}; P'' \Box c\{5\}; P'''$. Furthermore, punctual offers cannot be specified adequately: with $a; P \Box \mathbf{i}; \mathbf{stop}$, unlike $a\{0\}; P$, there is an uncertainty on the occurrence of a even if the environment is ready to interact at time 0.

Real-Time CSP [DaS94, BDS94]

The history of Real-Time CSP is already long. A detailed presentation of the language can be found in [Dav93]. We refer here to the most recent version (to our knowledge). Unlike some early proposals, arbitrary delays are no longer introduced between consecutive actions of a sequential process. However, the language still relies on a denotational model which requires that processes be time-guarded, i.e. there must be a strictly positive delay before any recursive instantiation of a process.

Real-Time CSP proposes an independent delay operator, and a nondeterministic choice which allows an easy definition of nondeterministic delays. The main difference with ET-LOTOS is in fact that internal actions do not resolve choices in CSP. Special operators are introduced to model time-outs and watchdogs. The behaviour of the time-out is indeed the same as a specification with an urgent **i**, in the sense that there is an uncertainty about what happens at the expiration of the time-out. Hence, as for Timed CCS above, this makes it impossible to specify adequately punctual offers. For this reason, a life reducer has been added to the last, and according to the authors, definitive version of the language [BDS94]. This final version has also been extended with $@t$.

Notice finally that our design of ET-LOTOS has been much inspired by an early version of Real-Time CSP [ReR88], which explains the similarities between the two formalisms.

The next three proposals are based on a same model. They were initially inspired by the second one: ATP. All three are restricted to a discrete time domain. The passing of one time unit is modelled by the occurrence of a special action σ (which we will call the "tick" action in the following): $P \xrightarrow{\sigma} P'$ means that P idles for one time unit and is thereby transformed into P'. They also share the same basic operator: $\lfloor P \rfloor(Q)$, called "unit delay" in ATP and "then"; in [HeR95]. It acts like a time-out of one time unit: either P performs an action immediately and the process behaves further like P alone, or P idles and after one time unit the process behaves like Q. We illustrate the expressing possibilities offered by this operator below.

TPL [HeR95] **and Temporal LOTOS** [Reg93]

TPL is an extension of CCS and Temporal LOTOS a version of TPL tailored towards LOTOS. Observable action prefix lets time pass indefinitely but maximal progress is applied: internal actions are urgent.

With the "then" operator, many mechanisms can already be described. A time-out of duration d for example, simply by combining several single "then":

$$\lfloor P \rfloor^{d+1}(Q) := \lfloor P \rfloor(\lfloor P \rfloor^{d}(Q))$$
$$\lfloor P \rfloor_0(Q) := \lfloor P \rfloor(Q)$$

or an independent delay, denoted $\sigma^n : \sigma^n Q \coloneqq [\mathbf{stop}]^n(Q)$.

Note that in the above time-out, P does not age. It is also possible however to model a time-out with an urgent **i**. Similarly, Temporal LOTOS can take advantage of the LOTOS disabling operator to model a watchdog.

ATP [NiS94]

The main difference with respect to TPL and Temporal LOTOS is that action prefix is urgent in ATP. Some operators are used to loosen this urgency when necessary. On the other hand, ATP is not based on the maximal progress property and there is no operator that allows one to enforce the occurrence of interactions. ATP also proposes other operators: a generalised time-out, a watchdog…

TPCCS [Han91]

This calculus extends CCS with time and probabilities. The timed extension is similar to both TPL and ATP. As in TPL, TPCCS has the maximal progress property and observable action prefix does not block time. It shares the definition of various additional operators (generalised time-out, watchdog…) with ATP.

The last four formalisms we present now differ from each other on many points. However, they also share some common features. First, none of them has the maximal progress property, but all propose urgent observable action prefixes. Also, they all include, under different versions, a "weak" choice, i.e. a choice which can also be resolved by idling.

Timed CCS [Che93]

In this calculus proposed by Liang Chen, also called Timed CCS like Wang Yi's proposal, no restriction is made on the time domain which may be discrete or dense. An independent delay operator, denoted (d), is proposed. Action prefix is enhanced with a time interval restricting the time at which the action may occur, and with a time measurement facility, similar to $@t : a^{e_1}_{e_2}(t); P$. e_1 and e_2 may be mathematical expressions (built with " + ", " − ", "$\underline{\min}$" and "$\underline{\max}$") on the time variables. A difference with ET-LOTOS is that an action prefix may not idle beyond the upper bound of the associated time interval (which may be ∞ however).

This urgency is loosened by the fact that the choice operator, denoted " + ", is weak: in $P+Q$ the choice can be made by idling when either P or Q blocks time. This can be used to model a simple time-out mechanism, e.g.: $a^5_0; P+(5); Q$ either performs a in the next 5 time units or behaves like Q after time 5. It is not clear however how a more general time-out operator on processes, e.g. $P \Box \Delta^d \mathbf{i}; Q$ in ET-LOTOS, could be defined with such a weak choice.

TCCS [MoT90]

TCCS also accepts a dense or a discrete time domain. Action-prefix is urgent but an operator, δ, is used to loosen this urgency: δP may perform the same actions as P at time 0, but it is ready to idle indefinitely before performing these actions. TCCS also includes two choice operators: a "strong" one à la ET-LOTOS and a weak one à la Timed CCS and an independent delay.

ACP_ρ[BaB91], **ACP_{drt} and ACP_{art}** [BaB96]

ACP_ρ is an extension of ACP with dense time. A time stamp is added to each action, indicating its occurrence time. Two theories are proposed in [BaB91], depending on whether the time stamps are taken to represent absolute or relative time. For example, $a(3).b(5)$ means "perform a at time 3 and b at time 5" (absolute time) and $a[3].b[5]$ means "perform a at time 3 and b 5 time units after a" (relative

time, which is indeed the classical interpretation followed by the other formalisms). Time cannot pass beyond the enabling time of an action. As in Chen's Timed CCS, however, the choice operator is weak. Another interesting operator is the integration, denoted by $\int_{v \in V} P$, where V is a set of positive real numbers. Intuitively, this is equivalent to a choice over values in V, except that the choice is weak: $\int_{v \in V} P$ may idle as long as there is at least one t in V such that $[t/v]P$ may idle. For example, $\int_{v \in [d, d']} a[v].P$ is equivalent to $a_d^{d'}(v).P$ in Chen's Timed CCS. The integration operator is more expressive, however, as V may be any set of positive reals and not just a simple interval.

ACP_{drt} is an extension of ACP with discrete time, using a special "tick" action like the other similar extensions. Actions are not decorated with time stamps anymore. They let time pass indefinitely, but an operator, **cts**(a), is used to make an action urgent. An independent delay of one time unit is introduced and the choice operator is still a weak one. A version of the theory with absolute time, ACP_{art}, is also presented.

$ACP^t_{t\varepsilon}$ [Gro90a]

$ACP^t_{t\varepsilon}$ is an extension of ACP with discrete time. The "tick" action denoted t, has the particularity to resolve the choices. For example $t.P + t.Q$ may idle and thereby be transformed into either P, or Q. Hence, among the formalisms we have presented, $ACP^t_{t\varepsilon}$ is the only one without the time determinism property. A direct consequence is that nondeterministic delays can be easily specified. For example: $P := t; P + t; a; Q$ can delay indefinitely the proposal of a. This requires however an extensive use of recursion.

Algebras for timed automata [WPD94, DaB96]

We finish this chapter with the presentation of two formalisms which are somehow an intermediate stage between timed process algebras and timed automata [AlD94]. Both are defined as "algebras for timed automata" and propose extensions of classical process algebras with clocks. In [WPD94] an extension of CCS is proposed. The principle is the same as for timed automata (see Section 8.6 below): a set of clocks is used to specify the timing constraints. The processes may test the values of the clocks by comparing them with integer constants and may reset the clocks. For this purpose, the action-prefix of CCS is extended and written $(\gamma, a, \Phi)\ P$. Φ indicates the set of clocks to be reset at the occurrence of the action a. γ is a boolean formula composed of atomic expressions of the form $x \mathrm{p} n$ for $\mathrm{p} \in \{<, >, \leqslant, \geqslant\}$, where x is a clock variable and n a natural number. The action a is enabled only when the clock values make γ true. Furthermore, a notion of necessity is associated with the action-prefix: $(\gamma, a, \Phi)\ P$ may not idle beyond the last time γ is verified. However, the choice operator is a weak one.

[DaB96] presents a similar proposal, also based on CCS but the parallel operator is the one of LOTOS. Another difference is that the resetting of the clocks is not linked to the edges of the automaton, i.e. to the occurrence of the transitions, but to the states. The language is also enhanced with three new constructs. The semantics of the calculus is given in terms of timed automata. An operational semantics translating each process into a LTS is also provided.

None of these two calculi has the maximal progress property and they do not consider the handling of data. The proposal of [WPD94], is a kind of minimal extension. It is interesting to note that the new action-prefix which is proposed is very similar to the combination of $@t$ and the selection predicate. Besides the urgency, the main difference is that $@t$ somehow declares a "temporary" clock

which is stopped (and thus disappears from the semantics) when the corresponding action occurs.

8.6. Other Timed Models

Up to now, we have focused mainly on process algebras which can be directly compared with our proposal. Although it is not the purpose of this paper to provide a complete survey of all the other proposed timed models, we will briefly mention some other very important ones.

Hybrid automata are generalized finite state machines extended with continuous variables. The discrete actions of a system are modelled by instantaneous transitions between a finite set of control locations (or states). The continuous activities of the environment are modelled by real valued variables which change continuously over time according to differential equations. Invariants on states specify conditions which must be verified by the variables to stay in these control locations. Transitions are controlled by predicates and their occurrence may change variable values. An important subclass of hybrid automata is the class of rectangular hybrid automata [Kop96] in which differential equations, invariants and transition predicates are represented by rectangles. An N-dimensional rectangle defines a set of points, represented by intervals of real numbers with rational end points for each dimension.

Timed Automata [AlD94] is a subclass of hybrid automata where variables are clocks. A clock is a positive variable that evolves uniformly with time. There are different variants of timed automata depending on the expressions allowed in predicates. Also, [SiY96] proposes Timed Automata with deadlines, where deadline conditions are associated with transitions rather than with states. They can be translated into standard timed automata, but their advantage is that they improve compositionality.

There are also several timed extensions of 1-safe Petri Nets with timed transitions or timed places, but these are special cases of Time Stream Petri Nets [SDS94, SiY96]. In this model, time intervals are associated with arcs and specify times when tokens become available to fire the corresponding transition.

TCTL [ACD90] is a timed extension of the temporal logic CTL which allows one to reason on timed systems states.

9. Conclusions

In this paper we have given a detailed presentation of the semantic model of ET-LOTOS, which is the basis of the timing extension of LOTOS included in E-LOTOS. In particular, we have especially taken care of proving the consistency of our semantics. We have also shown that it enjoys some nice properties: time is deterministic and additive, strong bisimulation is a congruence and ET-LOTOS is a conservative extension of LOTOS for guarded processes. To our knowledge, this is the first in-depth study of a language that combines data types and real-time behaviours.

It should be noted that our semantics does not resort to auxiliary functions and does not require any decoration of the labels of the LTS with additional parameters, nor the introduction of some new special actions besides the time values.

Defining a language and its theory is just a first step, which allows us to produce formal, unambiguous descriptions in ET-LOTOS. However, the main interest of

using our language lies in the automatic computations we could perform on the specifications. This requires of course the development of appropriate tools and beforehand, of an adequate theoretical framework.

In this respect, the translation of ET-LOTOS to timed automata open very interesting perspectives. The idea is of course to define a mapping between ET-LOTOS and a timed automaton, allowing the former to benefit from the model-checking theory and tools developed for the latter. Such a mapping has already been defined in [NSY92] for a version of ATP accepting a dense time domain. Then, the KRONOS tool [DOT+96] can be used. It takes a timed automaton and a TCTL formula as input and checks whether the formula is verified on the automaton. In [DOY95] a similar method is proposed for a subset of ET-LOTOS.

More recently, this work has been extended to cope with a larger subset of ET-LOTOS. [Her98] proposes an hybrid automaton model, called ETL-automaton, which is especially designed to support ET-LOTOS. Informally, it can be seen as a timed automaton extended with memory cells and ASAP (as soon as possible) transitions. The values of the memory cells and clocks can be used in guards to constrain the occurrence of transitions. Each state has an invariant condition, which must be verified for the automaton to stay in it. A transition can reset clocks and change the memory cells. In particular, it is possible to capture the clock values in memory cells, which makes it possible to capture the occurrence time of a transition in a variable, i.e. to model the $@t$ operator. The transitions are labelled either with a LOTOS action (i.e. an **i** or an observable gate with a list of attributes) or with a third special action marking the expiration of a delay. Each transition is also associated with a predicate on the clocks and the variables, which constrains its occurrence and determines the possible values of the attributes when the label is an observable action.

This ETL-automaton model covers nearly all the features of ET-LOTOS. The restrictions merely ensure the finiteness of the resulting automaton (e.g. no recursion through the parallel and the left part of the enabling and disabling operators) or ease the translation process (e.g. no recursion through the hiding operator, nor unguarded recursions through the delay or the guard operators).

A simulator of ETL-automata has been developed, which thus supports a very large subset of the language. [Her98] also studies how ETL-automata can be mapped onto underlying models of existing model-checkers such as HyTech [HHW95], KRONOS [DOT+96] and UPPAAL [LPY97]. Although the hybrid automata accepted by HyTech are the most general ones among these three, they are less expressive than ETL-automata, so that further restrictions should be considered. Basically, the ET-LOTOS expressions used in delays, life-reducers, selection predicates, offers, etc. must be linear, and the hide operator can only be used on non time-restricted actions. This still covers a large subset of the language, and the semi-decidable algorithm of HyTech can be used for reachability analysis. As regards KRONOS, its more restricted timed automaton model, motivated by decidability purposes, requires further restrictions. Basically, it seems difficult to model the time capture operator of ET-LOTOS because there are no memory cells. However, [Her97] shows that an extension of timed automata with semi-timers remains decidable and can support this operator. A timer is a clock that can be stopped and restarted, and a semi-timer is always reset before being restarted. The interesting feature is that a semi-timer can be modelled by two auxiliary ordinary clocks, in fact by their difference, so that, at the price of a larger number of clocks, the time-capture operator of ET-LOTOS can be supported by the KRONOS timed automata. Anyway, other restrictions exist: expressions in delays and life-reducers

should be constants, and expressions in selection predicates and guards are restricted. The same conclusion applies to UPPAAL, but the supported subset of ET-LOTOS is slightly larger, especially as regards the expressions in selection predicates and guards.

Finally, a denotational semantics of ET-LOTOS is proposed in [Bry97].

Acknowledgements

We are grateful to T. Bolognesi, J. Bryans, J.-P. Courtiat, J. Davies, A. Jeffrey, L. Llana, C. Pecheur, J. Quemada, G. Rabay, T. Regan, S. Schneider and S. Yovine for several interesting discussions on ET-LOTOS and its predecessors. Steve Schneider deserves special thanks for pointing out an error in a draft version of this paper. We are also grateful to anonymous referees for their judicious comments.

References

[ACD90] Alur, R., Courcoubetis, C. and Dill, D.: *Model-checking for real-time systems*. In IEEE 5th Annual Symposium on Logic In Computer Science (LICS), Philadelphia, June 1990.

[AlD94] Alur, R., Dill, D. L.: A Theory of Timed Automata. *Theoretical Computer Science*, 1994, 126, 183–235.

[BaB91] Baeten, J. C. M., Bergstra, J. A.: Real Time Process Algebra. *Formal Aspects of Computing* 3 (2) 142–188, 1991.

[BaB96] Baeten, J. C. M., Bergstra, J. A.: Discrete Time Process Algebra. *Formal Aspects of Computing* 8 (2) 188–208, 1996.

[BaV93] Baeten, J. and Verhoef, C.: *A Congruence Theorem for Structured Operational Semantics with Predicates*. In E. Best, ed, Proc. CONCUR 93, LNCS 715. Springer-Verlag, Berlin, 1993, 477–492.

[BDS94] Bryans, J., Davies, J., Schneider, S.: *Real-time CSP, and its relationship to ET-LOTOS*. Internal report, Univ. of Reading and Royal Holloway Univ. of London, 1994.

[BeK84] Bergstra, J. A., Klop, J. W.: Process Algebra for Synchronous Communication. *Information and Control* 37 (1985) 109–137.

[BIM95] Bloom, B., Istrail, S., Meyer, A. R.: Bisimulation can't be traced. *Journal of the ACM* 42, 1, 1995, 232–268.

[BLT94a] Bolognesi, T., Lucidi, F., Trigila, S.: Converging Towards a Timed LOTOS Standard. *Computer Standards and Interfaces*, 1994, 87–118.

[BLT94b] Bolognesi, T., Lucidi, F., Trigila, S.: *A Timed Full LOTOS with Time/Action Tree Semantics*. In: T. Rus, C. Rattray, eds., Theories and Experiences for Real-Time System Development, Amast Series in Computing Word Scientific, 1994, 205–237.

[BoB87] Bolognesi, T., Brinksma, E.: Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems* 14 (1) 25–59, 1987.

[BoG96] Bol, R. and Groote, J. F.: The Meaning of Negative Premises in Transition System Specifications. *Journal of the ACM*, 43(5): 863–914, 1996.

[Bri89] Brinksma, E.: *Constraint-oriented Specification in a Constructive Formal Description Technique*. In J. W. de Bakker, W.-P. de Roever, G. Rozenberg, eds., Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness, LNCS 430, Springer-Verlag, Berlin, 1990, 130–152.

[Bry97] Bryans, J. W.: *Denotational Semantics Models for Real-Time LOTOS*. Ph.D. Thesis. Reading University, UK.

[CdO94] Courtiat, J-P., de Oliveira, R.: *About time nondeterminism and exception handling in a temporal extension of LOTOS*. In: S. Vuong, S. Chanson, eds, Protocol Specification, Testing and Verification, XIV. Chapman & Hall, London, 1994, 37–52.

[Che93] Chen, L.: *Timed Processes: Models, Axioms and Decidability*. Ph.D Thesis, University of Edinburgh, Dept. of Computer Science, Edinburgh, Scotland.

[DaB96] D'Argenio, P. R., Brinksma, E.: *A Calculus for Timed Automata*. In: B. Jonsson and J. Parrow (eds.), Proc. of the 4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems, FTRTFT '96, Uppsala, Sweden, LNCS 1135, Springer Verlag, Berlin, 1996, 110–129.

[DaS94] Davies, J., Schneider, S.: *Real-time CSP*. In: T. Rus, C. Rathay, Eds., Theories and Experiences for Real-Time System Development, Amast Series in Computing, Word Scientific, 1994.

[Dav93] Davies, J.: *Specification and Proof in Real-Time CSP*. Cambridge University Press, 1993.

[DBS96] Davies, J., Bryans, J. W., Schneider, S.: *Real-Time LOTOS and Timed Observations*. In: G. von Bochmann, R. Dssouli, O. Rafiq, eds., Formal Description Techniques, VIII. Chapman & Hall, London, 1996.

[DOT+96] Daws, C., Olivero, A., Tripakis, S. and Yovine, S.: *The tool KRONOS*. In Hybrid Systems III, Verification and Control, LNCS 1066. Springer Verlag, Berlin, 1996.

[DOY95] Daws, C., Olivero, A., Yovine, S.: *Verifying ET-LOTOS programs with KRONOS*. In: D. Hogrefe, S. Leue, ed., Formal Description Techniques, VII. North-Holland, Amsterdam, 1995, 227–242.

[Gro90a] Groote, J. F.: *Specification and Verification of Real Time Systems in ACP*. In: L. Logrippo, R. Probert, H. Ural, eds., Protocol Specification, Testing and Verification, X, North-Holland, Amsterdam, 1990, 261–274.

[Gro 90b] Groote, J. F.: *Transition system specifications with negative premises*. In: J. C. M. Baeten, J. W. Klop, eds., CONCUR '90, Theories of Concurrency: Unification and Extension, LNCS 458. Springer-Verlag, Berlin, 1990, 332–341.

[Gro93] Groote, J. F.: *Transition system specifications with negative premises*. In: Theoretical Computer Science 118. Elsevier, 1993, 263–299.

[GrV92] Groote, J. F. and Vaandrager, F.: Structured Operational Semantics and Bisimulation as a Congruence. *Information and Computation*, 100, 2, 201–260, 1992.

[Han91] Hansson, H.: *Time and Probability in Formal Design of Distributed Systems*. Ph.D Thesis, DoCS 91/27, Uppsala University, Dept. of Computer Science, Uppsala, Sweden.

[HeR95] Hennessy, M., Regan, T.: A Process Algebra for Timed Systems. *Information and Computation* 117, 221–239 (1995).

[Her97] Hernalsteen, C.: *A Timed Automaton Model for ET-LOTOS Verification*. In: T. Mizuno, N. Shiratori, T. Higashino and A. Togashi, eds., Formal Description Techniques X and Protocol Specification, Testing and Verification XVII. Chapman & Hall, London, 1997, 193–204.

[Her98] Hernalsteen, C.: *Specification, Validation and Verification of Real-time Systems in ET-LOTOS*. Doctoral thesis, Free University of Brussels, June 1998.

[HHW95] Henzinger, T., Ho, P.-H. and Wong-Toi, H.: *A user guide to HyTech*. In K. Larsen, T. Margaria, E. Brinksma, W. Cleaveland and B. Steffen, eds., TACAS '95: Tools and Algorithms for the Construction and Analysis of Systems, LNCS 1019 (Springer Verlag, 1995), 41–71.

[ISO8807] ISO/IEC-JTC1/SC21/WG1/FDT/C. *LOTOS, a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*. IS 8807, Feb. 1989.

[ISO95] ISO/IEC JTC1/SC21/WG7. *Revised Working Draft on Enhancements to LOTOS*. (v2). Annex C of ISO/IEC JTC1/SC21/WG7 N1053, 1995.

[ISO98] ISO/IEC JTC1/SC33. *ISO/IEC FCD 15437 – Enhancements to LOTOS* (*E-LOTOS*). ISO/IEC JTC1/SC33 N0188, May 98, 209 p.

[KOP96] Kopke, P.: *The theory of rectangular hybrid automata*. PhD thesis, Faculty of the Graduate School of Cornell University, 1996.

[Led86] Leduc, G.: *Assessing the Service Provided by a Connection-less Protocol*. In: M. Diaz, ed., Protocol Specification, Testing and Verification, V. North-Holland, Amsterdam, 1986, 215–234.

[LeL93] Leduc, G., Léonard, L.: *A timed LOTOS supporting a dense time domain and including new timed operators*. In: M. Diaz, R. Groz, eds., Formal Description Techniques, V. North-Holland, Amsterdam, 1993, 87–102.

[LéL94] Léonard, L., Leduc, G.: *An Enhanced Version of Timed LOTOS and its Application to a Case Study*. In: R. Tenney, P. Amer, Ü. Uyar, eds., Formal Description Techniques, VI. North-Holland, Amsterdam, 1994, 483–498.

[LéL97] Léonard, L., Leduc, G.: An Introduction to ET-LOTOS for the Description of Time-Sensitive Systems. *Computer Networks and ISDN Systems* 29 (3), 1997, 271–292.

[Léo97] Léonard, L.: *An Extended LOTOS for the Design of Time-Sensitive Systems*. PhD Thesis, Université de Liège, Institut Montefiore B28, B-4000 Liège, 1, Belgique, April 1997.

[LLD94] Léonard, L., Leduc, G., Danthine, A.: *The Tick-Tock case study for the assessment of Timed FDTs*. In A. Danthine, ed., The OSI95 Transport Service with Multimedia Support. Springer-Verlag, Berlin, 1994, 338–352.

[LPY97] Larsen, K., Pettersson, P. and Yi Wang: Uppaal in a nutshell. Springer International Journal of Software Tools for Technology Transfer, 1 (1/2), Oct. 1997.

[Mil89] Milner, R.: *Communication and Concurrency*. Prentice-Hall International, London, 1989.
[MFV94] Miguel, C., Fernández, A., Vidaller, L.: *Extended LOTOS*. In: A. Danthine, ed., The OSI95 Transport Service with Multimedia Support. Springer-Verlag, Berlin, 1994, 312–337.
[MoT90] Moller, F., Tofts, C.: *A temporal calculus of communicating systems*. In: J. C. M. Baeten, J. W. Klop, eds., CONCUR '90, Theories of Concurrency: Unification and Extension, LNCS 458. Springer-Verlag, Berlin, 1990, 401–415.
[MPW92] Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes – I/II. *Information and Computation* 100, 1–40 & 41–77, 1992.
[MRV92] de Meer, J., Roth, R., Vuong, S.: Introduction to the Algebraic Specifications Based on the Language ACT ONE. *Computer Network and ISDN Systems* 23, 363–392, 1992.
[NiS92] Nicollin, X., Sifakis, J.: *An Overview and Synthesis on Timed Process Algebras*. In: K. G. Larsen, A. Skou, eds., Computer-Aided Verification, III, LNCS 575. Springer-Verlag, 1992, 376–398.
[NiS94] Nicollin, X., Sifakis, J.: *ATP:* Theory and Application. *Information and Computation* 114, 131–178, 1994.
[NOS93] Nicollin, X., Olivero, A., Sifakis, J. and Yovine, S.: *An Approach to the description and Analysis of Hybrid Systems*. In: Workshop on Theory of Hybrid Systems, LNCS 736. Springer-Verlag, Berlin, 1992, 149–178.
[Plo81] Plotkin, G. D.: *A Structural Approach to Operational Semantics*. Report DAIMI FN-19, Computer Science Dpt., Aarhus University, 1981.
[QMF94] Quemada, J., Miguel, C., de Frutos, D., Llana, L.: *A Timed LOTOS extension*. In: T. Rus, C. Rattray, eds., Theories and Experiences for Real-Time System Development, Amast Series in Computing (Word Scientific, 1994), 239–263.
[Reg93] Regan, T.: *Multimedia in Temporal LOTOS: a Lip-Synchronization Algorithm*. In: A. Danthine, G. Leduc, P. Wolper, eds., Protocol Specification, Testing and Verification, XIII. North-Holland, Amsterdam, 1993, 127–142.
[ReR88] Reed, G. M., Roscoe, A. W.: A Timed Model for Communicating Sequential Processes. *Theoretical Computer Science* 58, 249–261, 1998.
[SiY96] Sifakis, J. and Yovine, S.: *Compositional specification of timed systems*. In Proc. of the 13th Annual Symposium on Theoretical Aspects of Computer Science, STACS '96, LNCS 1046. Springer-Verlag, Berlin, 1996, 347–359.
[SDS94] Sénac, P., Diaz, M. and de Saqui-Sannes, P.: Toward a formal specification of multimedia scenarios. *Annals of Telecommunications*, 49 (5–6), 297–314, 1994.
[Tan96] Tanenbaum, A. S.: *Computer Networks*. Englewood Cliffs, NJ: Prentice Hall, 1996.
[Ver95] Verhoef, C.: A Congruence Theorem for Structured Operational Semantics with Predicates and Negative Premises. *Nordic Journal of Computing*, 2: 274–302, 1995.
[Wan91] Wang, Y.: *CCS + Time = an Interleaving Model for Real Time Systems*. In: J. Leach Albert, B. Mounier, M. Rodríguez Artalego, eds., Automata, Languages and Programming, 18, LNCS 510. Springer-Verlag, Berlin, 1991, 217–228.
[WPD94] Wang, Y., Pettersson, P. and Daniels, M.: *Automatic Verification of Real-Time Communicating Systems by Constraint-Solving*. In: D. Hogrefe, S. Leue, ed., Formal Description Techniques, VII. Chapman & Hall, London, 1995, 243–258.

Annex A1: Overview of the LOTOS operators

- **stop** is an inactive (deadlocked) process.
- $go_1 \ldots o_n[SP]; P$ (action-prefixing) is a process that first performs an (observable) action on gate g and then behaves like P. The tuple $o_1 \ldots o_n$ determines the data exchanged during the synchronisation: either data sent, by $!tx$, or data (of sort s) received, by $?x:s$. The variables declared in $o_1 \ldots o_n$ to receive data can appear in the selection predicate (i.e. the boolean expression) SP. Data can be received only if they verify SP.
- $\mathbf{i}; P$ is a process that first performs an internal action and then behaves like P.
- **exit**$(e_1, \ldots e_n)$ is a process that successfully terminates. It performs an action on gate δ and then turns into **stop**. The tuple $e_1, \ldots e_n$ determines the data transmitted to the subsequent process (see the enabling operator).

- $P_1 \Box P_2$ (choice) is a process that can behave either like P_1 or like P_2. The choice is resolved by the first process which performs an action. Notice that internal actions also resolve the choice.
- $P_1 \,|[\Gamma]|\, P_2$ is the parallel composition of P_1 and P_2 with synchronisation on the gates in Γ.
- **hide** Γ **in** P hides actions of P occurring at gates present in the set Γ, i.e. renames them **i**.
- $P_1 >>$ **accept** $x_1{:}s_1, \ldots x_n{:}s_n$ **in** P_2 (enabling) is the sequential composition of P_1 and P_2, i.e. P_2 can start when P_1 has terminated successfully. A process terminating successfully can transmit data to its successor: the tuple $e_1, \ldots e_n$ associated with **exit** determines the data values transmitted and **accept** $x_1{:}s_1, \ldots x_n{:}s_n$ **in** specifies the data P_2 expects to receive.
- $P_1 [> P_2$ (disabling) allows P_2 to disable P_1 provided P_1 has not terminated successfully.
- $[SP] \rightarrow P$ (guard) behaves like P if the guard SP is true and like **stop** otherwise.
- **let** $x_1 = tx_1, \ldots x_n = tx_n$ **in** P (instantiation) instantiates the free variables $x_1 \ldots x_n$ in P.
- **choice** $x_1{:}s_1, \ldots x_n{:}s_n \Box P$ (choice over values). Assuming P depends on the variables $x_1 \ldots x_n$, (of sorts $s_1 \ldots s_n$), **choice** $x_1{:}s_1, \ldots x_n{:}s_n \Box P$ offers a choice between the processes $P(tx_1 \ldots tx_n)$ for all the combinations of values $(tx_1 \ldots tx_n)$ of sorts $(s_1 \ldots s_n)$. For example, **choice** $x{:}Nat \Box P(x)$ means $P(0) \Box P(1) \Box P(2) \Box \ldots$

Annex A2: Semantics of datatypes in ACT ONE

Datatypes are represented by a *canonical algebraic specification CAS*, i.e. an algebraic specification $\langle S, OP, E \rangle$ (S is a set of sorts, OP is a set of operations and E is the set of conditional equations defined on the signature $\langle S, OP \rangle$) such that the signature $\langle S, OP \rangle$ contains all sorts and operations occurring in the behaviour part.

A data expression is called a term of this algebra. More precisely, a *term* is a composition of operations of OP which respects the sorts of arguments and results. In general, terms may contain variables. They are called *ground terms* if they don't.

The interpretation of CAS requires the generation of a *derivation system*, denoted DS. This derivation system is composed of axioms and inference rules generated by the conditional equations of E.

A congruence relation between ground terms is induced by CAS: two ground terms t_1 and t_2 are called *congruent* w.r.t. CAS, simply denoted $t_1 = t_2$, iff $DS \vdash t_1 = t_2$, i.e. it is possible to prove $t_1 = t_2$ from the axioms and the inference rules of the derivation system DS. In the sequel, we use a simplified notation, $\vdash t_1 = t_2$ to mean $DS \vdash t_1 = t_2$.

$[t]$ denotes the set of all ground terms congruent to t w.r.t. CAS, i.e. intuitively $[t]$ is the object represented by t or any of its equivalent representations.

The semantic interpretation of $CAS = \langle S, OP, E \rangle$ is the many-sorted algebra: $\Theta(CAS) = \langle D_Q, O_Q \rangle$, called the *quotient term algebra*, where

- D_Q is the set $\{\Theta(s) \mid s \in S\}$,
  where $\Theta(s) = \{[t] \mid t$ is a ground term of sort $s\}$ for each $s \in S$; and
- O_Q is the set of operations $\{\Theta(op) \mid op \in OP\}$,
  where the $\Theta(op)$ are defined by $\Theta(op)\,([t_1], \ldots [t_n]) = [op\,(t_1, \ldots t_n)]$.

In this algebra, the terms with different representations but modelling the same object are collapsed.

Annex A3: Proofs of propositions 6.1, 7.1 and 7.2

This annex is organised as follows. In section A3.1 we present some simple lemmas about *RE* that we will use in the following proofs. In section A3.2, we prove four lemmas. Three of them correspond to proposition 7.2 (time determinism) and to the two implications of proposition 7.1 (time additivity). The fourth one is necessary for the proof that *RE* verifies condition 1 (see section 6.3.3), which we give in section A3.3.

A3.1. Basic properties

Let us point out some properties of *RE*. They give an insight into the structure of *RE* and they will help simplify our reasoning in the sequel.

We recall hereafter the definition of *RE*.

$$RE = \bigcup_{0 \leqslant i < \aleph 1} RE_i$$

where[1]:

$$RE_i = \left\{ \phi \mid \phi \notin RE_{<i} \wedge \left(\exists \frac{\{\chi_k \mid k \in K\}}{\chi} \in R \, \exists \sigma \bullet \sigma(\chi) = \phi \wedge (\forall k \in K \bullet RE \models_i \sigma(\chi_k)) \right) \right\}$$

$RE \models_i \sigma(\chi_k)$ iff

$$\left([\sigma(\chi_k) \text{ is positive} \Rightarrow \sigma(\chi_k) \in RE_{<i}] \wedge [\sigma(\chi_k) = P \overset{g}{\not\rightarrow} \Rightarrow (\forall P' \forall a \in OA \bullet \underline{\text{name}}(a) = g \Rightarrow P \overset{a}{\rightarrow} P' \notin RE_{<i})] \right)$$

Notice that the condition on ϕ: "$\phi \notin RE_{<i}$", is not mandatory. As a consequence however, each transition only belongs to at most one RE_i[2], a property which will help simplify our reasoning in the sequel:

Property A3.1: $\phi \in RE \Leftrightarrow (\exists ! i < \aleph_1 \bullet \phi \in RE_i)$
(the notation $\exists !$ means: "there is one and only one").

Proof

$\Rightarrow$

$(\exists ! \, i < \aleph_1 \bullet \phi \in RE_i) \Rightarrow \phi \in RE$ is trivially true by definition of *RE*.

$\Leftarrow$

$\phi \in RE \Rightarrow (\exists i < \aleph_1 \bullet \phi \in RE_i)$ by definition of *RE*. Furthermore, it is not possible to have $\phi \in RE_i$ and $\phi \in RE_j$ with $j \neq i$. If $j > i$, by definition of RE_j: $\phi \notin RE_j$. And vice versa. □

---

[1] Remember that $RE_{<i} = \bigcup_{0 \leqslant j < i} RE_j$. In the sequel, we also use the notation: $RE_{\leqslant i} = \bigcup_{0 \leqslant j \leqslant i} RE_j$.

[2] Without this condition, a transition would also belong to all the following RE_i. Furthermore, a given transition can be derived from more than one set of premises. Consider for example: **i**; **stop**□**exit** >> **stop** $\overset{\mathbf{i}}{\rightarrow}$ **stop**. This transition could be derived by rule (Ch1) from **i**; **stop** $\overset{\mathbf{i}}{\rightarrow}$ **stop**, or by rule (Ch1′) from **exit** >> **stop** $\overset{\mathbf{i}}{\rightarrow}$ **stop**. Note however that **i**; **stop** $\overset{\mathbf{i}}{\rightarrow}$ **stop** $\in RE_0$, but **exit** >> **stop** $\overset{\mathbf{i}}{\rightarrow}$ **stop** $\in RE_1$.

In the sequel, we will call "layer" the RE_i's composing RE, and we will talk about the "level" of a transition in RE to refer to the indice of the layer the transition belongs to: "the level of ϕ is i" means $\phi \in RE_i$.

Property A3.2

$$\forall \frac{\{\chi_k \mid k \in K\}}{\chi} \in R \forall \sigma \forall l < \aleph_1 \bullet \sigma(\chi) = \phi \wedge (\forall k \in K \bullet RE \models_l \sigma(\chi_k)) \Rightarrow (\exists i \leqslant l \bullet \phi \in RE_i)$$

If all the premises required to derive a transition from a given rule are verified at a level i, then the transition necessarily belongs to RE_i or to a lower layer.

Proof

Either $\exists\, i < l \bullet \phi \in RE_i$ or $\phi \notin RE_{<l}$. In this last case, by definition of RE: $\sigma(\chi) = \phi \wedge (\forall k \in K \bullet RE \models_l \sigma(\chi_k)) \Rightarrow \phi \in RE_l$ □

A direct consequence of property A3.2 is that the level of a transition is the lowest possible one, i.e. the first one encompassing a sufficient set of premises.

A3.2. Propositions 7.1 and 7.2

Consider the following four lemmas:

Lemma A3.1: $\forall P, P', P'' \forall d \in D_{0\infty} \bullet P \xrightarrow{d} P' \wedge P \xrightarrow{d} P'' \Rightarrow P' = P''$

Lemma A3.2: $\forall\ P, P', P'' \forall d, d_1 \in D_{0\infty}\ \forall i < \aleph_1 \bullet$

$$P \xrightarrow{d+d_1} P'' \in RE_i \Rightarrow$$

$$(\exists P' \exists j, k < \aleph_1 \bullet P \xrightarrow{d} P' \in RE_j \wedge P' \xrightarrow{d_1} P'' \in RE_k \wedge j \leqslant i \wedge k \leqslant i)$$

Lemma A3.3: $\forall P, P', P'' \forall\, \alpha \in A \forall d \in D_{0\infty}\ \forall d_1 \in [0, d) \forall\, i, j < \aleph_1 \bullet$

$$P \xrightarrow{d} P' \in RE_i \wedge P_{(d_1)} \xrightarrow{\alpha} P'' \in RE_j \Rightarrow i \geqslant j$$

Lemma A3.4: $\forall P, P', P'' \forall d, d_1 \in D_{0\infty} \bullet P \xrightarrow{d} P' \wedge P' \xrightarrow{d_1} P'' \Rightarrow P \xrightarrow{d+d_1} P''$

Lemma A3.1 establishes the time determinism property (proposition 3.3).

Lemmas A3.2 and A3.4 correspond to the two implications of proposition 3.2. Moreover, lemma A3.2 provides additional information about the levels of the time transitions, which helps simplify the proof.

There is no real intuition behind lemma A3.3, but it expresses the fundamental property of our semantics which will allow us to prove that RE verifies condition 1. It states that the level of a transition $P \xrightarrow{d} P'$ is greater or equal to the level of any action transition P can perform immediately or at anytime within the next d time units (d not included). Notice that we use the shorthand notation $P_{(d)}$ in the definition of lemma A3.3 to avoid cumbersome notations. By lemma A3.1 and A3.2, we know that $P \xrightarrow{d} P'$ implies $\forall d_1 \in [0, d)$, $P_{(d_1)}$ exists and is unique.

We give hereafter the proofs of these four lemmas. It happens that the proof of lemma A3.3 uses lemma A3.2 which uses lemma A3.1. This explains why the lemmas are presented in that order.

Preliminary comments about the proofs

Notice that the four lemmas are of the form: $P \xrightarrow{d} P' \wedge \ldots \Rightarrow \ldots$. Hence, in the four cases we will proceed by induction upon the level of the transition $P \xrightarrow{d} P'$.[1]

Therefore, we will make an extensive use of the following reasoning.

Remember that $P \xrightarrow{\nu} P' \in RE_i$ implies by definition of RE_i that:

$$\exists \frac{\{\chi_k \mid k \in K\}}{\chi} \in R \exists \sigma \bullet \sigma(\chi) = P \xrightarrow{\nu} P' \wedge \ldots$$

i.e. $P \xrightarrow{\nu} P'$ is the conclusion of a rule in R whose premises are verified on $RE_{<i}$. As a matter of fact, there is in general just one rule in R (at most two) a given transition can be derived from. For example $\mathbf{stop} \Box \mathbf{exit} \xrightarrow{5} \mathbf{stop} \Box \mathbf{exit}$ is necessarily obtained from rule (Ch2). Moreover, for every rule in R except (GC1) and (GC2), only one set of premises can lead to a given conclusion (all the processes appearing in the premises also appear in the conclusion, or are fixed by the context: the condition $Q[g_1, \ldots g_n](x_1, \ldots x_m) := P$ in rule (In2)).

So, in the following proofs we will frequently say that a transition "implies" its premises because it is clear that they are the only ones the transition can be derived from. Hence, if the transition is of level i, we deduce from the definition of RE that for each premise χ: $RE \models_i \chi$. For example: $\mathbf{stop} \Box \mathbf{exit} \xrightarrow{5} \mathbf{stop} \Box \mathbf{exit} \in RE_i$ implies that for some $j, k < i$, $\mathbf{stop} \xrightarrow{5} \mathbf{stop} \in RE_j$ and $\mathbf{exit} \xrightarrow{5} \mathbf{exit} \in RE_k$.

Thanks to property A3.2, we know even more about i, j and k: necessarily, $i = \underline{\max}(j, k) + 1$. We have $i > \underline{\max}(j, k)$ as j and $k < i$. On the other hand, $RE \models_{\underline{\max}(j,k)+1} \mathbf{stop} \xrightarrow{5} \mathbf{stop}$ and $RE \models_{\underline{\max}(,k)+1} \mathbf{exit} \xrightarrow{5} \mathbf{exit}$. Hence, by property A3.2: $i \leqslant \underline{\max}(j, k) + 1$. We will frequently use property A3.2 in a similar way, without mentioning it when the result is obvious.

Finally, notice that the function "$-$" on time values used in the sequel is defined as in chapter 3 (3.6.4).

Proof of lemma A3.1: $\forall P, P', P'' \forall d \in D_{0\infty} \bullet P \xrightarrow{d} P' \wedge P \xrightarrow{d} P'' \Rightarrow P' = P''$

We proceed by induction upon the level of $P \xrightarrow{d} P'$.

It can be verified easily that knowing P and d suffices to determine the rule in R $P \xrightarrow{d} P'$ is derived from. So, we argue by case on the rule from which $P \xrightarrow{d} P'$ is derived:

Case 1: Rules (S), (AP2), (I2), (D3), (Ex2), (Ex3), (GC3), (G3).
All these rules are axioms, hence base cases in our induction procedure. It can be verified easily that all these rules are deterministic, i.e. in $P \xrightarrow{d} P'$, the value of P' is fixed by P and d.

Case 2: Rule (GC3).
We do not need induction in this case either.
(GC3) is not an axiom but given $P = \mathbf{Achoice}(d_1)x{:}s \Box Q$ and given d there is only one possible conclusion:

$$\mathbf{Achoice}(d_1)x{:}s \Box Q \xrightarrow{d} \mathbf{Achoice}(d + d_1)x{:}s \Box Q.$$

[1] The induction principle on the ordinal numbers can be expressed (where $X(i)$ is any formula with i as free variable):

$$\frac{\forall i < \aleph_1 \bullet (\forall j < i \bullet X(j)) \Rightarrow X(i)}{\forall i < \aleph_1 \bullet X(i)}$$

Case 3: All the other rules.

All the other rules are of the form: $\dfrac{P \xrightarrow{d} P', \ldots}{OP(P) \xrightarrow{d} OP'(P')}$ or $\dfrac{P \xrightarrow{d} P', Q \xrightarrow{d} Q'}{OP(P,Q) \xrightarrow{d} OP(P',Q')}$

where OP is an ET-LOTOS operator and $OP'(P') = OP(P')$ or P'. In all these cases, by induction, the proof is obvious. Let us study rule (Ch2) for example:

$P \Box Q \xrightarrow{d} P' \Box Q'$ is derived from $P \xrightarrow{d} P'$ and $Q \xrightarrow{d} Q'$ which are, then, of lower level. Hence we may apply our inductive hypothesis, and we find:

$\mathrm{P} \xrightarrow{d} P' \wedge P \xrightarrow{d} P'' \Rightarrow P' = P''$ and $Q \xrightarrow{d} Q' \wedge Q \xrightarrow{d} Q'' \Rightarrow Q' = Q''$.

So, $P \Box Q \xrightarrow{d} P'' \Box Q''$ being derived from $P \xrightarrow{d} P''$ and $Q \xrightarrow{d} Q''$, we have: $P' \Box Q' = P'' \Box Q''$. □

Proof of lemma A3.2: $\forall P,\ P',\ P''\forall\ d,\ d_1 \in D_{0\infty} \forall i < \aleph_1 \bullet$

$$P \xrightarrow{d+d_1} P'' \in RE_i \Rightarrow$$

$$(\exists P'\ \exists j,k < \aleph_1 \bullet P \xrightarrow{d} P' \in RE_j \wedge P' \xrightarrow{d1} P'' \in RE_k \wedge j \leqslant i \wedge k \leqslant i)$$

We proceed by induction upon the level of $P \xrightarrow{d+d_1} P''$. It is implicitly assumed in the sequel that d and d_1 are greater than *0*. We argue by case on the syntactic structure of P:

Case 1: $P = \mathbf{stop}$ $\quad P = go_1 \ldots o_n @ t[SP]; Q$ $\quad P = \mathbf{i} @ t\{d_2\}; Q$
$P = \mathbf{exit}(e_1, \ldots e_n)\{d_2\}$ $\quad P = [SP] \rightarrow Q$ with $\neg \vdash SP$.

These cases are base cases: $P \xrightarrow{d+d_1} P''$ is derived from an axiom.

Hence $P \xrightarrow{d+d_1} P'' \in RE_0$.
It is easy to verify that the lemma holds in all these cases.

Case 2: $P = \Delta^{d_2} Q$.

Depending on the values of d_2, d, d_1, we have to distinguish between different cases.

(i) $d + d_1 \leqslant d_2$

Then, $\Delta^{d_2} Q \xrightarrow{d+d_1} P''$ is necessarily derived from rule (D2) and $P'' = \Delta^{d_2-d-d_1} Q$. (D2) being an axiom, we have: $\Delta^{d_2} Q \xrightarrow{d+d_1} \Delta^{d_2-d-d_1} Q \in RE_0$

Similarly, it is straightforward by rule (D2) that: $\Delta^{d_2} Q \xrightarrow{d} \Delta^{d_2-d} Q \in RE_0$

and $\Delta^{d_2-d} Q \xrightarrow{d_1} \Delta^{d_2-d-d_1} Q \in RE_0$

(ii) $d < d_2$ but $d + d_1 > d_2$

Then, $\Delta^{d_2} Q \xrightarrow{d+d_1} P''$ is necessarily derived from rule (D3) and $\Delta^{d_2} Q \xrightarrow{d+d_1} P'' \in RE_i$ implies $Q \xrightarrow{d+d_1-d_2} P'' \in RE_{i-1}$.

So, as $d \leqslant d_2$, we have by rule (D2): $\Delta^{d_2} Q \xrightarrow{d} \Delta^{d_2-d} Q \in RE_0$

and by rule (D3), $Q \xrightarrow{d+d_1-d_2} P'' \in RE_{i-1}$ implies: $\Delta^{d_2-d} Q \xrightarrow{d_1} P'' \in RE_i$

(iii) $d > d_2$ (let $d = d_2 + d_3$, with $d_3 > 0$)

Then, $\Delta^{d_2} Q \xrightarrow{d+d_1} P''$ is necessarily derived from rule (D3) and

$$\Delta^{d_2} Q \xrightarrow{d+d_1} P'' \in RE_i$$

implies $Q \xrightarrow{d_3+d_1} P'' \in RE_{i-1}$
Applying our inductive hypothesis to $Q \xrightarrow{d_3+d_1} P''$, we find that for some Q':

$Q \xrightarrow{d_3} Q' \in RE_k$ and $Q' \xrightarrow{d_1} P'' \in RE_l$ with $k \leqslant i-1$ and $l \leqslant i-1$

So, by rule (D3): $Q \xrightarrow{d_3} Q' \in RE_k$ implies: $\Delta^{d_2} Q \xrightarrow{d} Q' \in RE_{k+1}$ with $k+1 \leqslant i$

and we have also: $Q' \xrightarrow{d_1} P'' \in RE_l$ with $l \leqslant i-1$

Case 3: $P = Q \Box R$.

Then, $Q \Box R \xrightarrow{d+d_1} P''$ is necessarily derived from rule (Ch2) and $Q \Box R \xrightarrow{d+d_1} Q'' \Box R'' \in RE_i$ implies $Q \xrightarrow{d+d_1} Q'' \in RE_k$ and $R \xrightarrow{d+d_1} R'' \in RE_l$ with $\underline{\max}(k,l) = i-1$
Applying our induction hypothesis, we have for some Q' and R':
$Q \xrightarrow{d} Q' \in RE_{k'}$ and $Q' \xrightarrow{d_1} Q'' \in RE_{k''}$ with $k' \leqslant k \leqslant i-1$ and $k'' \leqslant k \leqslant i-1$
and
$R \xrightarrow{d} R' \in RE_{l'}$ and $R' \xrightarrow{d_1} R'' \in RE_{l''}$ with $l' \leqslant l \leqslant i-1$ and $l'' \leqslant l \leqslant i-1$
So, by rule (CH2):
$Q \xrightarrow{d} Q' \in RE_{k'}$ and $R \xrightarrow{d} R' \in RE_{l'}$ imply: $Q \Box R \xrightarrow{d} Q' \Box R' \in RE_{kl'}$ with $kl' = \underline{\max}(k', l') + 1 \leqslant i$

and
$Q' \xrightarrow{d} Q'' \in RE_{k''}$ and $R' \xrightarrow{d} R'' \in RE_{l''}$ imply: $Q' \Box R' \xrightarrow{d} Q'' \Box R'' \in RE_{kl''}$ with $kl'' = \underline{\max}(k'', l'') + 1 \leqslant i$.

Case 4: $P = Q \,|[\Gamma]|\, R$ $\quad P = \mathbf{inf} \,|||\, Q$ $\quad P = Q [> R$
$P = [SP] \to Q$ with $\vdash SP$ $\quad P = \mathbf{let}\, x_1 = tx_1, \ldots x_n = tx_n \,\mathbf{in}\, Q$
$P = R[g_1, \ldots g_n](tx_1, \ldots tx_m)$ with $R[h_1, \ldots h_n](x_1, \ldots x_m) := Q$
All these cases are similar to case 3.
Case 5: $P = \mathbf{Achoice}(d_2)\, x{:}s \Box Q$.[1]
Then, $\mathbf{Achoice}(d_2)\, x{:}s \Box Q \xrightarrow{d+d_1} P''$ is necessarily derived from rule (GC3) and

$$\mathbf{Achoice}(d_2)\, x{:}s \Box Q \xrightarrow{d+d_1} \mathbf{Achoice}(d+d_1+d_2)\, x{:}s \Box Q \in RE_i$$

implies (with $T = \{t \,|\, [t] \in \Theta(s)\}$):
(1) $\forall\, tx \in T \exists Q' \exists l < \aleph_1 \bullet [tx/x]Q \xrightarrow{d+d_1+d_2} Q' \in RE_l \wedge l < i$
By induction, we may deduce:
(2) $\forall\, tx \in T \exists Q' \exists l, l' < \aleph_1 \bullet [tx/x]Q \xrightarrow{d+d_2} Q' \in RE_{l'} \wedge l' \leqslant l < i$
Then, by rule (GC3) and by application of property A3.2
(all the premises are verified at level i, then the conclusion belongs to a level $\leqslant i$),
(2) implies that for some k:

$$\mathbf{Achoice}(d_2)\, x{:}s \Box Q \xrightarrow{d} P' \; \mathbf{Achoice}(d+d_2)\, x{:}s \Box Q \in RE_k \text{ with } k \leqslant i$$

Similarly, by rule (GC3) and by application of property A3.2,

[1] To keep the notations simple, we will consider in the following, with no lack of generality, that the generalised choice operator only instantiates one variable: **choice** $x{:}s \Box Q$.

(1) implies that for some k':

$$\textbf{Achoice}(d+d_2)\,x{:}s\,\square\,Q \xrightarrow{d_1} P' \quad \textbf{Achoice}(d+d_1+d_2)\,x{:}s\,\square\,Q \in RE_{k'} \text{ with } k' \leqslant i$$

Case 6: $P = \textbf{hide}\ g\ \textbf{in}\ Q$

Then, $\textbf{hide}\ g\ \textbf{in}\ Q \xrightarrow{d+d_1} P''$ is necessarily derived from rule (H3) and

$$\textbf{hide}\ g\ \textbf{in}\ Q \xrightarrow{d+d_1} \textbf{hide}\ g\ \textbf{in}\ Q'' \in RE_i$$

implies:

(1) $Q \xrightarrow{d+d_1} Q'' \in RE_{<i}$

(2) $\forall P' \forall a \in OA \bullet \underline{\text{name}}(a) = g \Rightarrow Q \xrightarrow{a} P' \notin RE_{<i}$

(3) $\forall d_2 \in (0,\ d+d_1) \bullet (Q \xrightarrow{d_2} Q_{(d_2)} \in RE_{<i} \wedge$

$$(\forall P' \forall\, a \in OA \bullet \underline{\text{name}}(a) = g \Rightarrow Q_{(d_2)} \xrightarrow{a} P' \notin RE_{<i}))$$

Note that we use the notation $Q_{(d_2)}$ to recall that thanks to the time determinism property, the result of Q idling for d_2 time units is unique if it exists. In particular, Q'' might be denoted $Q_{(d+d_1)}$.

Note also that for every $d_2 \in (0, d+d_1)$ there is a $d_3 > 0$ such that $d_2 + d_3 = d + d_1$.

Hence, as $Q \xrightarrow{d+d_1} Q'' \in RE_{<i}$, we may apply our inductive hypothesis and we find:

(4) $\forall d_2, d_3 > 0 \bullet d_2 + d_3 = d + d_1 \Rightarrow Q \xrightarrow{d_2} Q_{(d_2)} \in RE_{<i} \wedge Q_{(d_2)} \xrightarrow{d_3} Q'' \in RE_{<i}$.

In particular:

$Q \xrightarrow{d} Q_{(d)} \in RE_{<i}$ and $Q_{(d)} \xrightarrow{d_1} Q'' \in RE_{<i}$.

So, we have:

$Q \xrightarrow{d} Q_{(d)} \in RE_{<i}$

and (2): $\forall P' \forall\, a \in OA \bullet \underline{\text{name}}(a) = g \Rightarrow Q \xrightarrow{a} P' \notin RE_{<i}$

and by (3): $\forall d_2 \in (0, d) \forall P'' \forall a \in OA \bullet \underline{\text{name}}(a) = g \Rightarrow Q_{(d_2)} \xrightarrow{a} P'' \notin RE_{<i}$

Hence, by rule (H3) and by property A3.2 (all the premises are verified at level i so the conclusion is at most at level i): $\textbf{hide}\ g\ \textbf{in}\ Q \xrightarrow{d} \textbf{hide}\ g\ \textbf{in}\ Q_{(d)} \in RE_{\leqslant i}$

It remains to prove that $\textbf{hide}\ g\ \textbf{in}\ Q_{(d)} \xrightarrow{d_1} \textbf{hide}\ g\ \textbf{in}\ Q'' \in RE_{\leqslant i}$.

We already know that $Q_{(d)} \xrightarrow{d_1} Q'' \in RE_{<i}$.

Hence $Q_{(d)} \xrightarrow{d_2} Q_{(d)(d_2)} \in RE_{<i}$ for every $d_2 \in (0, d_1)$.

To use property A3.2 with rule (H3), we still need to prove that:

(i) $\forall P' \forall\, a \in OA \bullet \underline{\text{name}}(a) = g \Rightarrow Q_{(d)} \xrightarrow{a} P' \notin RE_{<i}$

(ii) $\forall\, d_2 \in (0, d_1) \forall P'' \forall\, a \in OA \bullet \underline{\text{name}}(a) = g \Rightarrow Q_{(d)(d_2)} \xrightarrow{a} P'' \notin RE_{<i}$

Note that $Q \xrightarrow{d} Q_{(d)} \in RE_{<i}$. Hence, by (3), the proof of (i) is straightforward.

For the proof of (ii), let $d_2 \in (0, d_1)$. Then, $d + d_2 < d + d_1$ and by (4) we have:

$Q \xrightarrow{d+d_2} Q_{(d+d_2)} \in RE_{<i}$.

Applying once again our inductive hypothesis, we get:

$Q \xrightarrow{d} Q_{(d)} \in RE_{<i}$ (what we knew) and $Q_{(d)} \xrightarrow{d_2} Q_{(d+d_2)} \in RE_{<i}$.

Time transitions being deterministic, $Q_{(d)(d_2)} = Q_{(d+d_2)}$.

And by (3), we have: $\forall P' \forall\, a \in OA \bullet \underline{\text{name}}(a) = g \Rightarrow Q_{(d+d_2)} \xrightarrow{a} P' \notin RE_{<i}$

Case 7: $P = Q >> \textbf{accept}\ x_1{:}s_1 \ldots x_n{:}s_n\ \textbf{in}\ R$

Similar to case 6.□

Proof of lemma A3.3: $\forall P,\ P',\ P'' \forall\ \alpha \in A\ \forall d \in D_{0\infty}\ \forall\ d_1 \in [0, d) \forall\ i, j < \aleph_1 \bullet$

$$P \xrightarrow{d} P' \in RE_i \wedge P_{(d_1)} \xrightarrow{\alpha} P'' \in RE_j \Rightarrow i \geqslant j$$

We proceed by induction upon the level of $P \xrightarrow{d} P'$ and we argue by case on the syntactic structure of P.

Case 1: $P = \mathbf{stop}$
Whatever value of d, we have: $\mathbf{stop} \xrightarrow{d} \mathbf{stop} \in RE_0$ and **stop** may not perform any action. Hence the proof is trivial.

Case 2: $P = go_1 \ldots o_n@t[SP]; Q$
Whatever value of d, we have:
$$go_1 \ldots o_n@t[SP]; Q \xrightarrow{d} go_1 \ldots o_n@t[[t+d/t]SP]; [t+d/t]Q \in RE_0$$
And for the action transitions: $go_1 \ldots o_n@t[[t+d_1/t]SP]; [t+d_1/t]Q \xrightarrow{\alpha} P'' \in RE_0$
Hence, the proof is trivial.

Case 3: $P = \mathbf{i}\{d_2\}; Q$
Similar to case 2.

Case 4: $P = \Delta^{d_2} Q$
Depending on the values of d and d_2, $\Delta^{d_2} Q \xrightarrow{d} P'$ is derived either from rule (D2) or from rule (D3).
(i) If $d_2 \geqslant d$, rule (D2) is applied.
Let $d_2 = d + d_3, d_3 \geqslant 0$. According to (D2): $\Delta^{d_2} Q \xrightarrow{d} \Delta^{d_3} Q \in RE_0$.
Similarly, $\forall\, d_1 < d, \Delta^{d_2} Q \xrightarrow{d_1} \Delta^{d_4} Q$ with $d_4 > d_3$ and $\Delta^{d_4} Q$ may not perform any action.
Hence, the proof is trivial.
(ii) If $d_2 < d$, rule (D3) is applied.
Let $d = d_2 + d_3$, $d_3 > 0$. According to (D3): $\Delta^{d_2} Q \xrightarrow{d} P' \in RE_i$ implies $Q \xrightarrow{d_3} P' \in RE_{i-1}$.
So, we may apply our inductive hypothesis to $Q \xrightarrow{d_3} P'$ and we find:
(1) $\forall\, \alpha \in OA \forall d_1 \in [0, d_3) \bullet Q_{(d_1)} \xrightarrow{\alpha} P'' \in RE_j \Rightarrow j \leqslant i-1$
Now, let $d_1 < d$. We have to distinguish between different cases:
$d_1 + d_4 = d_2$ with $0 < d_4$ or $d_1 = d_2$ or $d_1 = d_2 + d_4$ with $0 < d_4 < d_3$
- If $d_1 + d_4 = d_2$, $\Delta^{d_2} Q \xrightarrow{d_1} \Delta^{d_4} Q$ with $d_4 > 0$.
  Hence, $\Delta^{d_4} Q$ cannot perform any action and the proof is trivial.
- If $d_1 = d_2$: $\Delta^{d_2} Q \xrightarrow{d_1} \Delta^0 Q$ and according to (D1):
  $\Delta^0 Q \xrightarrow{\alpha} P'' \in RE_j$ implies $Q \xrightarrow{\alpha} P'' \in RE_{j-1}$
  and we know by (1) that $Q \xrightarrow{\alpha} P'' \in RE_{j-1}$ implies $j-1 \leqslant i-1$.
- If $(d_1 = d_2 + d_4 \wedge 0 < d_4 < d_3)$: $\Delta^{d_2} Q \xrightarrow{d_1} Q_{(d_4)}$ and we know by (1) that $Q_{(d_4)} \xrightarrow{\alpha} P' \in RE_j$ implies $j \leqslant i-1$.

Case 5: $P = \mathbf{exit}(e_1 \ldots e_n)\{d_2\}$
Depending on the values of d and d_2, $\mathbf{exit}(e_1 \ldots e_n)\{d_2\} \xrightarrow{d} P'$ is derived either from rule (Ex2) or from rule (Ex3) and $P' = \mathbf{exit}\ (e_1 \ldots e_n)\{d_2 - d\}$ or $P' = \mathbf{stop}$.
stop may not perform any action and $\mathbf{exit}(e_1 \ldots e_n)\{d_2 - d\}$ may only perform actions transitions at level *0*, because (Ex1) is an axiom.
Hence $\forall\, d_1$ the level of $P_{(d_1)} \xrightarrow{\alpha} P''$ is at most *0* and the proof is trivial.

Case 6: $P = Q \Box R$
For the time transitions, only (Ch2) may be applied. Hence,
$$Q \Box R \xrightarrow{d} Q_{(d)} \Box R_{(d)} \in RE_i$$
implies $Q \xrightarrow{d} Q_{(d)} \in RE_j$ and $R \xrightarrow{d} R_{(d)} \in RE_k$, with $\underline{\max}(j, k) = i-1$
Then, by induction:
(1) $\forall\, d_1 \in [0, d) \forall \alpha \in OA \bullet Q_{(d_1)} \xrightarrow{\alpha} P'' \in RE_l \Rightarrow l \leqslant j < i$
(2) $\forall\, d_1 \in [0, d) \forall\, \alpha \in OA \bullet R_{(d_1)} \xrightarrow{\alpha} P'' \in RE_l \Rightarrow l \leqslant k < i$

Now, let $d_1 \in [0, d)$. Hence, $P_{(d_1)} = Q_{(d_1)} \Box R_{(d_1)}$.
Only (Ch1) and (Ch1′) may be applied to derive an action transition from $Q_{(d_1)} \Box R_{(d_1)}$.
If $Q_{(d_1)} \Box R_{(d_1)} \xrightarrow{\alpha} P'' \in RE_1$
is derived from (Ch1) then: $Q_{(d_1)} \xrightarrow{\alpha} P'' \in RE_{l-1}$
and we know by (1) that $l-1 < i$, i.e.: $l \leqslant i$.
The proof is the same with (Ch1′).

Case 7: $P = Q \,|[\Gamma]|\, R \quad P = \mathbf{inf} \,|||\, Q \quad P = Q \,[> R$
Similar to case 6.

Case 8: $P = [SP] \to Q$
Similar to case 1 if $\neg \vdash SP$.
If $\vdash SP$, $[SP] \to Q \xrightarrow{d} P'$ can only be derived from (G2).
So: $[SP] \to Q \xrightarrow{d} Q_{(d)} \in RE_i$
implies $Q \xrightarrow{d} Q_{(d)} \in RE_{i-1}$.
Then, we may apply our inductive hypothesis and we find:
(1) $Q_{(d_1)} \xrightarrow{\alpha} P'' \in RE_j \wedge d_1 \in [0, d) \Rightarrow j \leqslant i-1$
Now, let $d_1 \in [0, d)$. We have to distinguish between two cases: either $d_1 > 0$ or $d_1 = 0$.
If $d_1 > 0$, $P_{(d_1)} = Q_{(d_1)}$.
So, if $Q_{(d_1)} \xrightarrow{\alpha} P'' \in RE_j$
we know by (1) that $j \leqslant i-1$, i.e.: $j \leqslant i$
If $d_1 = 0$:
$P_0 = [SP] \to Q$ and according to (G1): $[SP] \to Q \xrightarrow{\alpha} P'' \in RE_j$
implies $Q \xrightarrow{\alpha} P'' \in RE_{j-1}$
and we know by (1) that $j-1 \leqslant i-1$, i.e.: $j \leqslant i$

Case 9: $P = \mathbf{let}\ x_1 = tx_1, \ldots x_n = tx_n\ \mathbf{in}\ Q$
$P = Q[g_1, \ldots g_n](tx_1, \ldots tx_m)$ with $Q[h_1, \ldots h_n](x_1, \ldots x_m) := R$
Similar to case 8 with $\vdash SP$.

Case 10: $P = \mathbf{Achoice}(d_2)\, x : s \Box Q$
For the time transitions, only (GC3) may be applied. Hence,
$\mathbf{Achoice}(d_2)\, x : s \Box\ Q \xrightarrow{d} \mathbf{Achoice}(d_2 + d)\, x : s \Box Q \in RE_i$
implies
(1) $\forall\, tx \in \Theta(s) \bullet ([tx/x]Q \xrightarrow{d_2+d} [tx/x]Q_{(d_2+d)} \in RE_{<i})$
Now, let $d_1 \in [0, d)$. Hence, $P_{(d_1)} = \mathbf{Achoice}(d_2 + d_1)\, x : s \Box Q$.
Two cases should be distinguished for $\mathbf{Achoice}(d_2 + d_1)\, x : s \Box Q \xrightarrow{\alpha} P''$.
(i) If $d_2 = 0$ and $d_1 = 0$, the transition is derived from rule (GC1).
Hence, $\mathbf{Achoice}(d_2 + d_1)\ x : s \Box Q \xrightarrow{\alpha} P'' \in RE_j$
implies that for some $tx \in \Theta(s)$: $[tx/x]Q \xrightarrow{\alpha} P'' \in RE_{j-1}$.
Furthermore, we know by (1) that: $[tx/x]Q \xrightarrow{d_2+d} [tx/x]Q_{(d_2+d)} \in RE_{<i}$.
Then, applying our inductive hypothesis, we find:
$[tx/x]Q \xrightarrow{\alpha} P'' \in RE_{j-1}$ implies $j-1 < i$, i.e.: $j \leqslant i$.
(ii) If $d_2 + d_1 > 0$, the transition is derived from rule (GC2).
Hence, $\mathbf{Achoice}\ (d_2 + d_1)\ x : s \Box Q \xrightarrow{\alpha} P'' \in RE_j$
implies that for some $tx \in \Theta(s)$: $[tx/x]Q \xrightarrow{d_2+d_1} [tx/x]Q_{(d_2+d_1)} \in RE_k$
and $[tx/x]Q_{(d_2+d_1)} \xrightarrow{\alpha} P'' \in RE_l$ with $\underline{\max}(k, l) = j-1$.
Furthermore, we know by (1) that: $[tx/x]Q \xrightarrow{d_2+d} [tx/x]Q_{(d_2+d)} \in RE_{<i}$
and we have: $d_2 + d_1 < d_2 + d$ as $d_1 < d$.

Then, applying our inductive hypothesis, we find:
$[tx/x]Q_{(d_2+d_1)} \xrightarrow{\alpha} P'' \in RE_{<i}$. Hence, $l < i$.
We may also apply lemma A3.2, and we find:
$[tx/x]Q \xrightarrow{d_2+d_1} [tx/x]Q_{(d_2+d_1)} \in RE_{<i}$. Hence, $k < i$.
So, $\underline{\max}(k, l) = j-1 < i$, i.e.: $j \leqslant i$.

Case 11: $P =$ **hide** g **in** Q
hide g **in** $Q \xrightarrow{d} P'$ can only be derived from rule (H3). Hence,
$$\textbf{hide } g \textbf{ in } Q \xrightarrow{d} \textbf{hide } g \textbf{ in } Q_{(d)} \in RE_i$$
implies:
(1) $Q \xrightarrow{d} Q_{(d)} \in RE_{i-1}$
(2) $\forall d_2 \in [0, d) \forall Q'' \forall a \in OA \bullet \underline{\text{name}}(a) = g \Rightarrow Q_{(d_2)} \xrightarrow{a} Q'' \notin RE_{<i}$
Now, let $d_1 \in [0, d)$. Hence, $P_{(d_1)} =$ **hide** g **in** $Q_{(d_1)}$.
A priori, **hide** g **in** $Q_{(d_1)} \xrightarrow{\alpha} P''$ could be derived either from (H1) or from (H2).
In fact, it cannot be derived from (H2). We prove it by contradiction.
Suppose that **hide** g **in** $Q_{(d_1)} \xrightarrow{\alpha} P''$ is derived from (H2).
So, $\alpha =$ **i** and $Q_{(d_1)} \xrightarrow{a} P''$ for some a with $\underline{\text{name}}(a) = g$, which contradicts (2).
Thus, **hide** g **in** $Q_{(d_1)} \xrightarrow{\alpha} P'' \in RE_j$ can only be derived from (H1), and we have:
$$\textbf{hide } g \textbf{ in } Q_{(d_1)} \xrightarrow{\alpha} P'' \in RE_j$$
implies $Q_{(d_1)} \xrightarrow{\alpha} P'' \in RE_{j-1}$
Then, as $Q \xrightarrow{d} Q_{(d)} \in RE_{i-1}$ and $d_1 < d$,
we find by induction $j-1 \leqslant i-1$, i.e.: $j \leqslant i$

Case 12: $P = Q >>$ **accept** $x_1 : s_1 \ldots x_n : s_n$ **in** R
Similar to case 10□

This last lemma makes the additivity property complete. Hence, we give its proof now. Nevertheless, it is not used in the proof that *RE* verifies condition 1. So, we will already assume that this last property is verified (this helps simplify our reasoning in the case of **hide**).

Proof of lemma A3.4: $\forall P, P', P'' \forall d, d_1 \in D_{0\infty} \bullet P \xrightarrow{d} P' \wedge P' \xrightarrow{d_1} P'' \Rightarrow P \xrightarrow{d+d_1} P''$

We proceed by induction upon the level of $P \xrightarrow{d} P'$ and we argue by case on the syntactic structure of P.

Case 1: $P =$ **stop**
This is a base case. By rule (S), $\forall\, d, d_1 > 0$ we have $\textbf{stop} \xrightarrow{d} \textbf{stop} \xrightarrow{d_1} \textbf{stop}$
and also: $\textbf{stop} \xrightarrow{d+d_1} \textbf{stop}$.

Case 2: $P = go_1 \ldots o_n@t[SP]; Q$
This case is also a base case.
By rule (AP2), $go_1 \ldots o_n@t[SP]; Q \xrightarrow{d} go_1 \ldots o_n@t[[t+d/t]SP]; [t+d/t]Q$
and $go_1 \ldots o_n@t[[t+d/t]SP]; [t+d/t]Q \xrightarrow{d_1} go_1 \ldots o_n@t[[t+d_1/t][t+d/t]SP]; [t+d_1/t][t+d/t]Q$.
And also:
$$go_1 \ldots o_n@t[SP]; Q \xrightarrow{d+d_1} go_1 \ldots o_n@t[[t+d+d_1/t]SP]; [t+d+d_1/t]Q.$$
The proof is done as we have:
$go_1 \ldots o_n@t[[t+d_1/t][t+d/t]SP]; [t+d_1/t][t+d/t]Q =$
$go_1 \ldots o_n@t[[t+d+d_1/t]SP]; [t+d+d_1/t]Q$.

Case 3: $P = \mathbf{i}\{d_2\}; Q$
Similar to case 2.

Case 4: $P = \Delta^{d_2} Q$
Depending on the values of d and d_1, we must consider different cases.
(i) $d + d_1 \leqslant d_2$.
Let $d + d_1 + d_3 = d_2$, with $d_3 \geqslant 0$.
Then, by rule (D2), $\Delta^{d_2} Q \xrightarrow{d} \Delta^{d_1+d_3} Q \xrightarrow{d_1} \Delta^{d_3} Q$.
And also by rule (D2), $\Delta^{d_2} Q \xrightarrow{d+d_1} \Delta^{d_3} Q$.
(ii) $d \leqslant d_2$ and $d + d_1 > d_2$.
Let $d + d_3 = d_2$, and $d + d_1 = d_2 + d_4$, with $d_3 \geqslant 0$ and $d_4 > 0$. (Hence $d_1 = d_3 + d_4$).
Then, by rule (D2), $\Delta^{d_2} Q \xrightarrow{d} \Delta^{d_3} Q$.
By rule (D3), $\Delta^{d_3} Q \xrightarrow{d_1} Q'$
implies that $Q \xrightarrow{d_4} Q'$.
Also, by rule (D3), $Q \xrightarrow{d_4} Q'$ implies that $\Delta^{d_2} Q \xrightarrow{d_4+d_2} Q'$ and $d_4 + d_2 = d + d_1$.
(iii) $d > d_2$.
Let $d = d_2 + d_3$, with $d_3 > 0$.
Then, by rule (D3), $\Delta^{d_2} Q \xrightarrow{d} Q'$
is derived from $Q \xrightarrow{d_3} Q'$.
Hence, $Q \xrightarrow{d_3} Q'$ is of lower level than $\Delta^{d_2} Q \xrightarrow{d} Q'$
and if $Q' \xrightarrow{d_1} Q''$
we may apply our induction hypothesis: $Q \xrightarrow{d_3} Q'$ and $Q' \xrightarrow{d_1} Q''$ imply $Q \xrightarrow{d_3+d_1} Q''$.
Finally, by rule (D3): $Q \xrightarrow{d_3+d_1} Q''$ implies $\Delta^{d_2} Q \xrightarrow{d_1+d_3+d_2} Q''$ with $d_1 + d_3 + d_2 = d + d_1$.

Case 5: $P = \mathbf{exit}(e_1 \ldots e_n)\{d_2\}$
Depending on the values of d and d_1, we must consider different cases.
(i) $d + d_1 \leqslant d_2$. Let $d + d_1 + d_3 = d_2$, with $d_3 \geqslant 0$.
Then, by rule (Ex2),
$\mathbf{exit}(e_1 \ldots e_n)\{d_2\} \xrightarrow{d} \mathbf{exit}(e_1 \ldots e_n)\{d_1 + d_3\} \xrightarrow{d_1} \mathbf{exit}(e_1 \ldots e_n)\{d_3\}$.
Also by rule (Ex2), $\mathbf{exit}(e_1 \ldots e_n)\{d_2\} \xrightarrow{d+d_1} \mathbf{exit}(e_1 \ldots e_n)\{d_3\}$.
(ii) $d \leqslant d_2$ and $d + d_1 > d_2$. Let $d + d_3 = d_2$, with $d_1 \geqslant d_3 \geqslant 0$.
Then, by rule (Ex2), $\mathbf{exit}(e_1 \ldots e_n)\{d_2\} \xrightarrow{d} \mathbf{exit}(e_1 \ldots e_n)\{d_3\}$.
By rule (Ex3), we have: $\mathbf{exit}(e_1 \ldots e_n)\{d_3\} \xrightarrow{d_1} \mathbf{stop}$.
Also, by rule (Ex3): $\mathbf{exit}(e_1 \ldots e_n)\{d_2\} \xrightarrow{d+d_1} \mathbf{stop}$.
(iii) $d > d_2$.
Then, by rule (Ex3): $\mathbf{exit}(e_1 \ldots e_n)\{d_2\} \xrightarrow{d} \mathbf{stop}$.
By rule (S), $\mathbf{stop} \xrightarrow{d_1} \mathbf{stop}$.
Also, by rule (Ex3), $\mathbf{exit}(e_1 \ldots e_n)\{d_2\} \xrightarrow{d+d_1} \mathbf{stop}$.

Case 6: $P = Q \Box R$
By rule (Ch2): $Q \Box R \xrightarrow{d} Q' \Box R' \xrightarrow{d_1} Q'' \Box R''$
implies $Q \xrightarrow{d} Q' \xrightarrow{d_1} Q''$ and $R \xrightarrow{d} R' \xrightarrow{d_1} R''$.
$Q \xrightarrow{d} Q'$ and $R \xrightarrow{d} R'$ are of lower level than $Q \Box R \xrightarrow{d} Q' \Box R'$.
So, we may apply our inductive hypothesis and we find:
$Q \xrightarrow{d+d_1} Q''$ and $R \xrightarrow{d+d_1} R''$.
Hence, by rule (Ch2): $Q \Box R \xrightarrow{d+d_1} Q'' \Box R''$

Case 7: $P = Q \,|[\Gamma]|\, R \quad P = \mathbf{inf} \,|||\, Q \quad P = Q\,[> R$
Similar to case 6.

Case 8: $P = [SP] \rightarrow Q$
Similar to case 1 if $\neg \vdash SP$.
If $\vdash SP$, by rule (G2), $[SP] \rightarrow Q \xrightarrow{d} Q'$

implies $Q \xrightarrow{d} Q'$.
Hence, the level of $Q \xrightarrow{d} Q'$ is lower than the level of $[SP] \rightarrow Q'$ and if: $Q' \xrightarrow{d_1} Q''$
we find by induction: $Q \xrightarrow{d} Q'$ and $Q' \xrightarrow{d_1} Q''$ imply $Q \xrightarrow{d_1+d} Q''$.
Hence, by rule (G2): $[SP] \rightarrow Q \xrightarrow{d_1+d} Q''$

Case 9: $P =$ **let** $x_1 = tx_1, \ldots x_n = tx_n$ **in** Q
$P = Q[g_1, \ldots g_n](tx_1, \ldots tx_m)$ with $Q[h_1, \ldots h_n](x_1, \ldots x_m) := R$
Similar to case 8 with $\vdash SP$.

Case 10: $P =$ **Achoice**$(d_2)\, x{:}s \square Q$
By rule (GC3), **Achoice**$(d_2)\, x{:}s \square Q \xrightarrow{d}$ **Achoice**$(d_2+d)\, x{:}s \square Q$
And also, **Achoice**$(d_2+d)\, x{:}s \square Q \xrightarrow{d_1}$ **Achoice**$(d_2+d+d_1)\, x{:}s \square Q$
implies: $\forall\, tx \in \Theta(s) \bullet [tx/x]Q \xrightarrow{d_1+d+d_2}$
So, all the premises required to derive

$$\textbf{Achoice}(d_2)\, x{:}s \square Q \xrightarrow{d_1+d+d_2} \textbf{Achoice}(d_2+d+d_1)\, x{:}s \square Q \in RE_{\leqslant i}$$

from rule (GC3) are verified. Hence, this transition belongs to RE as RE verifies condition 2.

Case 11: $P =$ **hide** g **in** Q
By (H3), as RE verifies condition 1: **hide** g **in** $Q \xrightarrow{d}$ **hide** g **in** Q'
implies
(1) $Q \xrightarrow{d} Q'$
(2) $\forall\, d_2 \in [0, d) \bullet Q_{(d_2)} \overset{g}{\not\rightarrow}$
Similarly, **hide** g **in** $Q' \xrightarrow{d_1}$ **hide** g **in** Q''
implies:
(3) $Q' \xrightarrow{d_1} Q''$
(4) $\forall\, d_2 \in [0, d_1) \bullet Q'_{(d_2)} \overset{g}{\not\rightarrow}$
The level of $Q \xrightarrow{d} Q'$ being lower than the one of **hide** g **in** $Q \xrightarrow{d}$ **hide** g **in** Q', we may apply our inductive hypothesis to: $Q \xrightarrow{d} Q'$ and $Q' \xrightarrow{d_1} Q''$.
Then, we find: (5) $Q \xrightarrow{d_1+d} Q''$
Similarly, $\forall\, d_2 \in [0, d_1)$: $Q \xrightarrow{d} Q'$ and $Q' \xrightarrow{d_2} Q'_{(d_2)}$ imply $Q \xrightarrow{d+d_2} Q'_{(d_2)}$.
Hence, we can group the results of (2) and (4) and we find:
(6) $\forall\, d_2 \in [0, d+d_1) \bullet Q'_{(d_2)} \overset{g}{\not\rightarrow}$
So, (5) and (6) imply that all the premises required to derive **hide** g **in** $Q \xrightarrow{d_1+d}$ **hide** g **in** Q'' from rule (H3) are verified.
Hence, this transition belongs to RE as RE verifies condition 2.

Case 12: $P = Q >>$ **accept** $x_1{:}s_1 \ldots x_n{:}s_n$ **in** R
Similar to case 10 □

A3.3. Proof that *RE* verifies condition 1 (of section 6.3.3)
We have to prove that any transition in RE can be derived from a rule in R whose premises are verified on RE. Formally:

$$\phi \in RE \Rightarrow (\exists \frac{\{\chi_k \mid k \in K\}}{\chi} \in R, \exists\, \sigma \bullet (\sigma(\chi) = \phi \wedge \forall k \in K \bullet RE \vDash \sigma(\chi_k)))$$

According to property A3.1 and to the definition of RE, this can be written:

$$(\exists!\, i < \aleph_1 \bullet \phi \notin RE_{<i} \wedge \exists \frac{\{\chi_k \mid k \in K\}}{\chi} \in R, \exists\, \sigma \bullet (\sigma(\chi) = \phi \wedge \forall k \in K \bullet RE \vDash_i \sigma(\chi)))$$

$$\Rightarrow \exists \frac{\{\chi_k \mid k \in K\}}{\chi} \in R, \exists\, \sigma \bullet (\sigma(\chi) = \phi \wedge \forall k \in K \bullet RE \vDash \sigma(\chi_k))$$

i.e. we know that a transition belongs to RE only if it can be derived from a rule in R whose premises are verified on $RE_{<i}$ for some i. Hence, it would be enough to prove:

$$\forall i < \aleph_1 \forall \frac{\{\chi_k \mid k \in K\}}{\chi} \in R \forall \sigma \bullet$$
$$((\forall k \in K \bullet RE \models_i \sigma(\chi_k)) \Rightarrow (\forall k \in K \bullet RE \models \sigma(\chi_k))) \qquad (*)$$

We will argue by case on the different rules in R.
Consider first all the rules which have positive premises only. Then, the proof is straightforward: if the positive premises belong to $RE_{<i}$, they necessarily belong to RE:
$\forall k \in K \bullet RE \models_i \sigma(\chi_k)$ implies:
(1) $\forall k \in K \bullet \sigma(\chi_k) \in RE_{<i}$
and clearly: (1) $\Rightarrow \forall k \in K \bullet \sigma(\chi_k) \in RE$

$\Rightarrow \forall k \in K \bullet RE \models \sigma(\chi_k)$

The difficulty comes indeed from the negative premises: a negative premise verified on $RE_{<i}$ does not necessarily remain so on RE, i.e. a transition falsifying it could be in a layer higher than i. Nevertheless, we will show that thanks to lemma A3.3, this never happens to be the case in ET-LOTOS.

Only two rules in R have negative premises: (H3) and (En3). We consider (H3) below. The proof for (En3) is similar.
So, let $\{\sigma(\chi_k) \mid k \in K\} = \{P \xrightarrow{d} P'\}$

$\cup \{P \overset{g}{\not\rightarrow} \mid g \in \Gamma\}$

$\cup \{P \xrightarrow{d_1} P'' \wedge P'' \overset{g}{\not\rightarrow} \mid g \in \Gamma, d_1 \in (0, d)\}$

Then, $\forall k \in K \bullet RE \models_i \sigma(\chi_k)$ implies:
(2) $P \xrightarrow{d} P' \in RE_{<i}$
(3) $\forall a \in OA \bullet (\underline{\text{name}}(a) \in \Gamma \Rightarrow \forall P'' \bullet P \xrightarrow{a} P'' \notin RE_{<i})$
(4) $\forall d_1 \in (0, d) \bullet \quad P \xrightarrow{d_1} P'' \in RE_{<i}$

$\wedge$

$\forall a \in OA \bullet (\underline{\text{name}}(a) \in \Gamma \Rightarrow \forall P''' \bullet P'' \xrightarrow{a} P''' \notin RE_{<i})$

And we have to prove:
(2′) $P \xrightarrow{d} P' \in RE$
(3′) $\forall a \in OA \bullet (\underline{\text{name}}(a) \in \Gamma \Rightarrow \forall P'' \bullet P \xrightarrow{a} P'' \notin RE)$
(4′) $\forall d_1 \in (0, d) \bullet \quad P \xrightarrow{d_1} P'' \in RE$

$\wedge$

$\forall a \in OA \bullet (\underline{\text{name}}(a) \in \Gamma \Rightarrow \forall P''' \bullet P'' \xrightarrow{a} P''' \notin RE)$

For (2′), the case is obvious: (2) $\Rightarrow$ (2′).
By lemma A3.3 and by (2), we also find:
(5) $\forall a \in OA \forall P'' \forall j < \aleph_1 \bullet P \xrightarrow{a} P'' \in RE_j \Rightarrow j < i$.
And:
(3) $\wedge$ (5)

$\Rightarrow \forall a \in OA \bullet (\underline{\text{name}}(a) \in \Gamma \Rightarrow \forall P'' \forall j < \aleph_1 \bullet P \xrightarrow{a} P'' \notin RE_{<i} \wedge (P \xrightarrow{a} P'' \in RE_j \Rightarrow j < i.))$
$\Rightarrow \forall a \in OA \bullet (\underline{\text{name}}(a) \in \Gamma \Rightarrow \forall P'' \forall j < \aleph_1 \bullet P \xrightarrow{a} P'' \notin RE_j)$
$\Rightarrow (3')$.
Similarly, (2) and lemma A3.3 imply:
(6) $\forall d_1 \in (0, d) \forall a \in OA \forall P''' \forall j < \aleph_1 \bullet P'' \xrightarrow{a} P''' \in RE_j \Rightarrow j < i.$
Hence: (4) $\wedge$ (6) $\wedge$ $(P \xrightarrow{d_1} P'' \in RE_{<i} \Rightarrow P \xrightarrow{d_1} P'' \in RE)$
$\Rightarrow$
$\forall d_1 \in (0, d) \forall P'' \bullet$

$$P \xrightarrow{d_1} P'' \in RE$$

$$\wedge$$

$$\forall a \in OA \bullet (\underline{\text{name}}(a) \in \Gamma \Rightarrow \forall P''' \bullet P'' \xrightarrow{a} P''' \notin RE_{<i} \wedge (P'' \xrightarrow{a} \mathrm{P}''' \in RE_j \Rightarrow j < i))$$

$\Rightarrow (4')$

Annex A4: Proofs of propositions 7.3 and 7.5

Proposition 7.3
Let $P \sim_e \mathrm{Q}$. Then for any of the 14 contexts defined in section 7.2, $C(P) \sim_e \mathrm{C(Q)}$.

Proof
Proposition 7.3 can be written:

$$P \sim_e \mathrm{Q} \Rightarrow \forall\, C(\bullet) \forall [/] \in \underline{\text{Sub}}(C(P), C(Q)) \bullet [/]C(P) \sim [/]C(Q)\ (*)$$

Notice first that some contexts $C(\bullet)$ may contain free data variables. In that case however, $[/]C(P) = [/]C'(P)$, where $C'(\bullet)$ is simply the context $C(\bullet)$ instantiated by $[/]$. So, we may restrict our proof to "closed" contexts only.

Then, for the first 9 contexts, $[/]\mathrm{C(P)} = \mathrm{C}([/]\mathrm{P})$ and $[/]C(Q) = C([/]Q)$, where $[/]P$ and $[/]Q$ are closed processes and $P \sim_e \mathrm{Q} \Rightarrow [/]P \sim [/]Q$.

For the last five contexts, $\underline{\text{Sub}}(C(P,C(Q))$ is not necessarily equal to $\underline{\text{Sub}}(P,Q)$. These contexts declare data variables that can be free in P or Q. Hence, $\underline{\text{Sub}}(C(P),C(Q))$ is a subset of $\underline{\text{Sub}}(P,Q)$. In this case, $[/]C(P)$ and $[/]C(Q)$ can be written $C([/]P)$ and $C([/]Q)$. We denote by $[/]$ a subset of $[/]$ from which the variables declared in the context are not substituted. So, $[/]P$ and $[/]Q$ are not closed, but $C([/]P)$ and $C([/]Q)$ are. Furthermore, $P \sim_e \mathrm{Q} \wedge [/]P \sim_e [/]Q$.

Hence, to prove $(*)$, it is sufficient to examine the cases where $C(P)$ and $C(Q)$ are closed and to show that $C(P) \sim C(Q)$ if $P \sim_e \mathrm{Q}$.

Let us define the relation $\mathscr{R} = \mathscr{R}1 \cup \mathscr{R}2 \cup \mathscr{R}3$ with:

$\mathscr{R}1 = \quad \{\langle P, Q\rangle \mid P \sim Q\}$
$\mathscr{R}2 = \quad \{\langle C(P), C(Q)\rangle \mid P \sim_e \mathrm{Q} \wedge C(P),\ C(Q)$ are closed behaviour expressions$\}$
$\mathscr{R}3 = \quad \{\langle P_1 \,|||\, \ldots \,|||\, P_n \,|||\, \mathbf{inf} \,|||\, P,\ Q_1 \,|||\, \ldots \,|||\, Q_n \,|||\, \mathbf{inf} \,|||\, Q\rangle \mid P \sim Q$ and $P_i \sim Q_i,\ i \leqslant n\}$

The proof that strong bisimulation is a congruence is done if we show that $\mathscr{R}$ is a strong bisimulation relation.

To prove that $\mathscr{R}$ is a strong bisimulation, we first show that all the couples in $\mathscr{R}2$ verify the transfer relation (by definition, this is true for the couples in $\mathscr{R}1$).

Proof for $\mathscr{R}2$

We prove that if $C(P)$, $C(Q)$ are closed behaviour expressions, with $P \tilde{e} Q$, then $C(P) \xrightarrow{\nu} P'$ implies that for some Q', $C(Q) \xrightarrow{\nu} Q'$ with $\langle P', Q' \rangle \in \mathscr{R}$.
We argue by case on the rule from which $C(P) \xrightarrow{\nu} P'$ is derived.
With the rules (D1, D3, Ch1, GC1, GC2, Di2 for $C6(\bullet)$, Di3 for $C5(\bullet)$, G1, G2, L1, L2):
$C(P) \xrightarrow{\nu} P'$ is derived from a premise of the form $P \xrightarrow{\nu} P'$ or $[/]P \xrightarrow{\nu} P'$.
Hence, for some $Q' : Q \xrightarrow{\nu} Q'$ or $[/]Q \xrightarrow{\nu} Q'$, with $P' \sim Q'$.
And by application of the same rule as for P: $C(Q) \xrightarrow{\nu} Q'$, with $\langle P', Q' \rangle \in \mathscr{R}1$.
With the rules (Ch2, PC1, PC2, PC3, IP2, H1, H2, En1 for $C7(\bullet)$, Di1 for $C5(\bullet)$, Di4):
$P' = C(P'')$ and P'' is determined by a premise of the form: $P \xrightarrow{\nu} P''$. (Some of these rules also have a second premise which depends on the context only, e.g. the premise $R \xrightarrow{d} R'$ in rule (Ch2) for the context $C(\bullet) = \{\bullet \Box R\}$).
Hence, for some $Q'' : Q \xrightarrow{\nu} Q''$, with $P'' \sim Q''$.
And by application of the same rule as for P: $C(Q) \xrightarrow{\nu} C(Q'')$, with $\langle C(P''), C(Q'') \rangle \in \mathscr{R}2$.
With the rules (PC1′, En1 for $C13(\bullet)$, En3 for $C13(\bullet)$, Di1 for $C6(\bullet)$):
$P' = C'(P)$, where $C'(\bullet)$ is determined by premises dependent on the context only.
Hence, we also have $C(Q) \xrightarrow{\nu} C'(Q)$, with $\langle C'(P), C'(Q) \rangle \in \mathscr{R}2$.
With the rules (CH1′, Di2 for $C5(\bullet)$, Di3 for $C6(\bullet)$, G3):
$P' = R'$, where R' is a closed expression determined by premises dependent on the context only.
Hence, we also have $C(Q) \xrightarrow{\nu} R'$, with $\langle R', R' \rangle \in \mathscr{R}1$.
With the rules (AP1, I1, En2 for $C13(\bullet)$):
$P' = [/]P$, where the instantiation is determined by ν.
Furthermore, (AP1) and (I1) are axioms and with (En2) for $C13(\bullet)$, the premise depends on the context only.
Hence, $C(P) \xrightarrow{\nu} [/]P$ implies $C(Q) \xrightarrow{\nu} [/]Q$.
And $P \tilde{e} Q$ implies $[/]P \sim [/]Q$, so that $\langle [/]P, [/]Q \rangle \in \mathscr{R}1$.
With the rule (H3) (the proof is similar with the rule (En3) for $C7(\bullet)$):
hide g **in** $P \xrightarrow{d}$ **hide** g **in** P' implies $P \xrightarrow{d} P'$ and $\forall d' < d \bullet P_{(d')} \xrightarrow{g}$.
$P \tilde{e} Q$ implies: $Q \xrightarrow{d} Q'$ with $P' \tilde{e} Q'$. And similarly, $\forall d' < d \bullet P_{(d')} \tilde{e} Q_{(d')}$.
Hence, $\forall d' < d \bullet Q_{(d')} \xrightarrow{g}$. So, **hide** g **in** $Q \xrightarrow{d}$ **hide** g **in** Q' and $\langle$**hide** g **in** P', **hide** g **in** $Q' \rangle \in \mathscr{R}2$.
Rule (D2) is an axiom.
So, $\Delta^{d_1+d} P \xrightarrow{d} \Delta^{d_1} P$ and $\Delta^{d_1+d} Q \xrightarrow{d} \Delta^{d_1} Q$.
And $\langle \Delta^{d_1} P, \Delta^{d_1} Q \rangle \in \mathscr{R}2$.
By rule (GC3):
Achoice$(d_1)\, x : s \Box P \xrightarrow{d}$ **Achoice**$(d_1 + d)\, x : s \Box P$ implies
$\forall tx \in \Theta(s) \bullet [tx/x]P \xrightarrow{d+d_1}$. Hence, $\forall tx \in \Theta(s) \bullet [tx/x]Q \xrightarrow{d+d_1}$.
And by (GC3): **Achoice**$(d_1)\, x : s \Box Q \xrightarrow{d}$ **Achoice**$(d_1 + d)\, x : s \Box Q$,
with $\langle$**Achoice**$(d_1 + d)\, x : s \Box P$, **Achoice**$(d_1 + d)\, x : s \Box Q \rangle \in \mathscr{R}2$.
By rule (En2 for $C7(\bullet)$):
$P >>$ **accept** $x_1 : s_1, \ldots x_n : s_n$ **in** $R \xrightarrow{\mathbf{i}} [v_1/x_1, \ldots v_n/x_n]R$ implies $P \xrightarrow{\delta v_1 \ldots v_n}$.
Hence, $Q \xrightarrow{\delta v_1 \ldots v_n}$.

And by (En2): $Q >>$ **accept** $x_1{:}s_1, \ldots x_n{:}s_n$ **in** $R \xrightarrow{\mathrm{i}} [v_1/x_1, \ldots v_n/x_n]R$,
with $\langle [v_1/x_1, \ldots v_n/x_n]R, [v_1/x_1, \ldots v_n/x_n]R \rangle \in \mathscr{R}1$.

Rule (AP2) is an axiom (the proof is similar for rule (I2)).

So, $go_1 \ldots o_n@t[SP]; P \xrightarrow{d} go_1 \ldots o_n@t[[t+d/t]SP]; [t+d/t]P$
and $go_1 \ldots o_n@t[SP]; Q \xrightarrow{d} go_1 \ldots o_n@t[[t+d/t]SP]; [t+d/t]Q$.
And $P \tilde{e} Q$ implies $[t+d/t]P \tilde{e} [t+d/t]Q$,
so that $\langle go_1 \ldots o_n@t[[t+d/t]SP]; [t+d/t]P, go_1 \ldots o_n@t[[t+d/t]SP]; [t+d/t]Q \rangle \in \mathscr{R}2$.

Finally, with (IP1):

inf $|||$ $P \xrightarrow{\alpha} P''$ $|||$ **inf** $|||$ P implies $P \xrightarrow{\alpha} P''$.
Hence, $Q \xrightarrow{\alpha} Q''$, with $P'' \sim Q''$.
And by (IP1): **inf** $|||$ $Q \xrightarrow{\alpha} Q''$ $|||$ **inf** $|||$ Q,
with $\langle P'' ||| \mathbf{inf} ||| Q, Q'' ||| \mathbf{inf} ||| Q \rangle \in \mathscr{R}3$. □

Proof for $\mathscr{R}3$

It remains to show that the couples in $\mathscr{R}3$ also verify the transfer relation.
The only case that is worth detailing is when **inf** $|||$ $P \xrightarrow{\alpha} P'$ $|||$ **inf** $|||$ P. Then $P \xrightarrow{\alpha} P'$ and consequently, $Q \xrightarrow{\alpha} Q'$, with $P' \sim Q'$, and **inf** $|||$ $Q \xrightarrow{\alpha} Q'$ $|||$ **inf** $|||$ Q.

Hence, $P_1 ||| \ldots ||| P_n ||| \mathbf{inf} ||| P \xrightarrow{\alpha} P_1 ||| \ldots ||| P_n ||| P' ||| \mathbf{inf} ||| P$
and $Q_1 ||| \ldots ||| Q_n ||| \mathbf{inf} ||| Q \xrightarrow{\alpha} Q_1 ||| \ldots ||| Q_n ||| Q' ||| \mathbf{inf} ||| Q$
and clearly: $\langle P_1 ||| \ldots ||| P_n ||| P' ||| \mathbf{inf} ||| P, Q_1 ||| \ldots ||| Q_n ||| Q' ||| \mathbf{inf} ||| Q \rangle \in \mathscr{R}3$. □

Proposition 7.5
$P \sim_{\mathrm{p}} \mathrm{Q} \Rightarrow R \sim S$ where $R := [R/X]P$, $S := [S/X]Q$.
($[R/X]P$ meaning that the variable X is replaced by the process name R)

Proof

We prove that when $P \sim_{\mathrm{p}} \mathrm{Q}$, the following relation:
$\mathscr{R} = \{\langle [R/X]G, [S/X]G \rangle |$ where G contains at most the process variable $X\}$ is a strong bisimulation.
For then, by taking $G = X$, it follows that $R \sim S$.
As shown in [Mil89], it is in fact sufficient to prove that $\mathscr{R}$ is a strong bisimulation up to $\sim$, i.e. every couple in $\mathscr{R}$ verifies the transfer relation up to $\sim$:

(i) if $[R/X]G \xrightarrow{\nu} R'$ then $\exists R'' \sim R' \exists S', S'' \bullet [S/X]G \xrightarrow{\nu} S' \sim S''$, with $\langle R'', S'' \rangle \in \mathscr{R}$
(ii) if $[S/X]G \xrightarrow{\nu} S'$ then $\exists S'' \sim S' \exists R', R'' \bullet [R/X]G \xrightarrow{\nu} R' \sim R''$, with $\langle R'', S'' \rangle \in \mathscr{R}$.

The following lemma will be useful:

Lemma A4.1

Let $R := [R/X]P, S := [S/X]Q$, with $P \sim_{\mathrm{p}} \mathrm{Q}$.
Then: $\forall d \in D_{0\infty} \forall \alpha \in A \bullet [R/X]G \xrightarrow{d} \wedge [S/X]G \xrightarrow{d} \Rightarrow \{[R/X]G \xrightarrow{\alpha} \Leftrightarrow [S/X]G \xrightarrow{\alpha}\}$.

Proof

By symmetry, it is enough to show that:
$[R/X]G \xrightarrow{d} \wedge [S/X]G \xrightarrow{d} \Rightarrow ([R/X]G \xrightarrow{\alpha} \Rightarrow [S/X]G \xrightarrow{\alpha})$.
We will proceed by transfinite induction upon the maximum of the levels of $[R/X]G \xrightarrow{d}$ and $[\mathrm{S/X}]\mathrm{G} \xrightarrow{d}$. Remember (see Annex A3) that the level of a transition is the index of the layer the transition belongs to in RE. As in the

proofs of Annex A3 we write below that a transition "implies" other transitions, meaning that the latters are the only possible premises the former can be derived from. Hence, the positive premises have a lower level than their conclusion.

Case 1: $G = X$.
Then, $[R/X]G = R$ and $[S/X]G = S$.
$R \xrightarrow{d}$ and $S \xrightarrow{d}$ imply: $[R/X]P \xrightarrow{d}$ and $[S/X]Q \xrightarrow{d}$.
$R \xrightarrow{\alpha}$ implies: $[R/X]P \xrightarrow{\alpha}$.
Hence, by induction: $[S/X]P \xrightarrow{\alpha}$.
But $P \sim_{\mathrm{p}} \mathrm{Q}$, so $[S/X]Q \xrightarrow{\alpha}$,
and then: $S \xrightarrow{\alpha}$.

Case 2: $G = go_1 \ldots o_n@t[SP]; G_1$.
Then, $[R/X]G = go_1 \ldots o_n@t[SP]; [R/X]G_1$
and $[S/X]G = go_1 \ldots o_n@t[SP]; [S/X]G_1$.
By (AP1), if $go_1 \ldots o_n@t[SP]; [R/X]G_1 \xrightarrow{gv_1 \ldots v_n} [v_1/o_1, \ldots v_m/o_m, 0/t][R/X]G_1$
then also: $go_1 \ldots o_n@t[SP]; [S/X]G_1 \xrightarrow{gv_1 \ldots v_n} [v_1/o_1, \ldots v_m/o_m, o/t][S/X]G_1$.

Case 3: $G = \mathbf{i}@t; G_1$.
Similar to the previous case.

Case 4: $G = \Delta^{d_1} G_1$.
Then, $[R/X]G = \Delta^{d_1}[R/X]G_1$
and $[S/X]G = \Delta^{d_1}[S/X]G_1$.
$\Delta^{d_1}[R/X]G_1$ can perform an action only if $d_1 = 0$.
$\Delta^0[R/X]G_1 \xrightarrow{d}$ and $\Delta^0[S/X]G_1 \xrightarrow{d}$ imply: $[R/X]G_1 \xrightarrow{d}$ and $[S/X]G_1 \xrightarrow{d}$.
$\Delta^0[R/X]G_1 \xrightarrow{\alpha}$ implies: $[R/X]G_1 \xrightarrow{\alpha}$.
Hence, by induction: $[S/X]G_1 \xrightarrow{\alpha}$,
and then: $\Delta^0[S/X]G_1 \xrightarrow{\alpha}$.

Case 5: $G = G_1 \Box G_2 \quad G = G_1 [> G_2$
These cases are simpler than the next one.

Case 6: $G = G_1 \,|[\Gamma]|\, G_2$.
Then, $[R/X]G = [R/X]G_1 \,|[\Gamma]|\, [R/X]G_2$
and $[S/X]G = [S/X]G_1 \,|[\Gamma]|\, [\mathrm{S/X}]G_2$.
$[R/X]G_1 \,|[\Gamma]|\, [R/X]G_2 \xrightarrow{d}$ implies: $[R/X]G_1 \xrightarrow{d}$ and $[R/X]G_2 \xrightarrow{d}$.
$[S/X]G_1 \,|[\Gamma]|\, [S/X]G_2 \xrightarrow{d}$ implies: $[S/X]G_1 \xrightarrow{d}$ and $[S/X]G_2 \xrightarrow{d}$.
Let $[R/X]G_1 \,|[\Gamma]|\, [R/X]G_2 \xrightarrow{\alpha}$. Depending on whether $\underline{\mathrm{name}}(\alpha) \in \Gamma$ or not, this action comes from the synchronisation of both processes or from the action of one of them only. We shall consider the case of the synchronisation only, which is more complete. So:
$[R/X]G_1 \,|[\Gamma]|\, [R/X]G_2 \xrightarrow{\alpha}$ implies: $[R/X]G_1 \xrightarrow{\alpha}$ and $[R/X]G_2 \xrightarrow{\alpha}$ with $\underline{\mathrm{name}}(\alpha) \in \Gamma$.
Hence, by induction: $[S/X]G_1 \xrightarrow{\alpha}$ and $[S/X]G_2 \xrightarrow{\alpha}$,
and then: $[S/X]G_1 \,|[\Gamma]|\, [S/X]G_2 \xrightarrow{\alpha}$.

Case 7: $G = \mathbf{Achoice}(d_1)\ x{:}s \Box G_1$.
Then, $[R/X]G = \mathbf{Achoice}(d_1)\ x{:}s \Box [R/X]G_1$
and $[S/X]G = \mathbf{Achoice}(d_1)\ x{:}s \Box [S/X]G_1$.
$\mathbf{Achoice}(d_1)\ x{:}s \Box [R/X]G_1 \xrightarrow{d}$ implies: $\forall tx \in \Theta(s) \bullet [tx/x][R/X]G_1 \xrightarrow{d+d_1}$.
$\mathbf{Achoice}(d_1)\ x{:}s \Box [S/X]G_1 \xrightarrow{d}$ implies: $\forall tx \in \Theta(s) \bullet [tx/x][S/X]G_1 \xrightarrow{d+d_1}$.
Let $\mathbf{Achoice}(d_1)\ x{:}s \Box [R/X]G_1 \xrightarrow{\alpha}$, with $d_1 > 0$. (The proof is simpler with $d_1 = 0$).
Then for some tx, $[tx/x][R/X]G_1 \xrightarrow{d_1} R_{1'}$ and $R_{1'} \xrightarrow{\alpha}$.

Note that by time additivity: $R_{1'} \xrightarrow{d}$ and similarly: $[tx/x][S/X]G_1 \xrightarrow{d_1} S_{1'}$ and $S_{1'} \xrightarrow{d}$.
Hence, by induction, we have $S_{1'} \xrightarrow{\alpha}$.
and then: **Achoice**(d_1) $x{:}s \square [S/X]G_1 \xrightarrow{\alpha}$.

Case 8: $G =$ **inf** $|||\, G_1$.
Similar to case 6.

Case 9: $G =$ **hide** g **in** G_1.
Then, $[R/X]G =$ **hide** g **in** $[R/X]G_1$
and $[S/X]G =$ **hide** g **in** $[S/X]G_1$.
hide g **in** $[R/X]G_1 \xrightarrow{d}$ implies: $[R/X]G_1 \xrightarrow{d}$.
hide g **in** $[S/X]G_1 \xrightarrow{d}$ implies: $[S/X]G_1 \xrightarrow{d}$.
Let **hide** g **in** $[R/X]G_1 \xrightarrow{\alpha}$ **hide** g **in** R'. Either this transition is derived from (H1) or (H2).
Consider that it is derived from (H2) (the demonstration is similar with (H1)).
Then, $\alpha =$ **i** and $[R/X]G_1 \xrightarrow{\alpha'}$ with $\underline{\text{name}}(\alpha') = g$.
Hence, by induction: $[S/X]G_1 \xrightarrow{\alpha'}$,
and then, **hide** g **in** $[S/X]G_1 \xrightarrow{\mathbf{i}}$.

Case 10: $G = G_1 >>$ **accept** $x_1{:}s_1, \ldots x_n{:}s_n$ **in** G_2.
Similar to case 9.

Case 11: $G = [SP] \to G_1$.
Then, $[R/X]G = [SP] \to [R/X]G_1$
and $[S/X]G = [SP] \to [S/X]G_1$.
$[SP] \to [R/X]G_1$ can perform an action only if $\vdash SP$.
In this case, the demonstration is similar to case 4 with $d_1 = 0$.

Case 12: $G =$ **let** $x_1 = tx_1, \ldots x_n = tx_n$ **in** G_1.
Similar to case 4 with $d_1 = 0$.

Case 13: $G = R_1[g_1, \ldots g_n](tx_1, \ldots tx_n)$ with $R_1[h_1, \ldots h_n](x_1, \ldots x_n) \coloneqq Q_1$.
$R_1[g_1, \ldots g_n](tx_1, \ldots tx_n)$ is a constant and does not contain any variable, so it is not influenced by the instantiation with R or S and the proof is immediate. □

To prove that all the couples in $\mathscr{R}$ verify the transfer relation up to $\sim$, it is sufficient to show that:

$$\text{if } [R/X]G \xrightarrow{\nu} R' \text{ then for some } S', S'': [S/X]G \xrightarrow{\nu} S'' \sim S', \text{ with } \langle R', S' \rangle \in \mathscr{R}.$$

We do our proof by induction upon the level of the transition $[R/X]G \xrightarrow{\nu} R'$.

Case 1: $G = X$.
Then, $[R/X]G = R$ and $[S/X]G = S$.
$R \xrightarrow{\nu} R'$ is obtained either by rule (In1) or (In2).
In both cases, $R \xrightarrow{\nu} R'$ implies that $[R/X]P \xrightarrow{\nu} R'$, by definition of R.
Hence, by induction, for some S', S'': $[S/X]P \xrightarrow{\nu} S'' \sim S'$, with $\langle R', S' \rangle \in \mathscr{R}$.
But $P \tilde{p} Q$, so $[S/X]Q \xrightarrow{\nu} S''' \sim S'$,
and by rule (In1) or (In2), $S \xrightarrow{\nu} S''' \sim S'$ with $\langle R', S' \rangle \in \mathscr{R}$.

Case 2: $G = go_1 \ldots o_n @ t[SP]; G_1$.
Then, $[R/X]G = go_1 \ldots o_n @ t[SP]; [R/X]G_1$
and $[S/X]G = go_1 \ldots o_n @ t[SP]; [S/X]G_1$.

If $go_1 \ldots o_n @ t[SP]; [R/X]G_1 \xrightarrow{gv_1 \ldots v_n} [v_1/o_1, \ldots v_m/o_m, 0/t][R/X]G_1$
then also: $go_1 \ldots o_n @ t[SP]; [S/X]G_1 \xrightarrow{gv_1 \ldots v_n} [v_1/o_1, \ldots v_m/o_m, 0/t][S/X]G_1$.
And clearly, $\langle [v_1/o_1, \ldots v_m/o_m, 0/t][R/X]G_1, [v_1/o_1, \ldots v_m/o_m, 0/t[S/X]G_1 \rangle \in \mathscr{R}$.
The proof is similar for the time transitions.

Case 3: $G = \mathbf{i}@t; G_1$.
Similar the previous case.

Case 4: $G = \Delta^{d_1} G_1$.
Then, $[R/X]G = \Delta^{d_1}[R/X]G_1$
and $[S/X]G = \Delta^{d_1}[S/X]G_1$.
If $\Delta^0[R/X]G_1 \xrightarrow{\alpha} R'$ (if d_1 is greater than 0, no action can be performed), then $[R/X]G_1 \xrightarrow{\alpha} R'$.
Hence, by induction, $[S/X]G_1 \xrightarrow{\alpha} S'' \sim S'$, with $\langle R', S' \rangle \in \mathscr{R}$
and by rule (D1): $\Delta^0[S/X]G_1 \xrightarrow{\alpha} S'' \sim S'$.
If $d_1 = d_2 + d$ with $d_2 \geqslant 0$, then $\Delta^{d_1}[R/X]G_1 \xrightarrow{d} \Delta^{d_2}[R/X]G_1$ and $\Delta^{d_1}[S/X]G_1 \xrightarrow{d} \Delta^{d_2}[S/X]G_1$.
And clearly, $\langle \Delta^{d_2}[R/X]G_1, \Delta^{d_2}[S/X]G_1 \rangle \in \mathscr{R}$.
If $d = d_1 + d_2$ with $d_2 > 0$, and $\Delta^{d_1}[R/X]G_1 \xrightarrow{d} R'$ then $[R/X]G_1 \xrightarrow{d_2} R'$.
Hence, by induction, $[S/X]G_1 \xrightarrow{d_2} S'' \sim S'$, with $\langle R', S' \rangle \in \mathscr{R}$
and by rule (D3): $\Delta^{d_1}[S/X]G_1 \xrightarrow{d_2} S'' \sim S'$.

Case 5: $G = G_1 \Box G_2 \qquad G = G_1 [> G_2$
These cases are simpler than the next one.

Case 6: $G = G_1 \,|[\Gamma]|\, G_2$.
Then, $[R/X]G = [R/X]G_1 \,|[\Gamma]|\, [R/X]G_2$
and $[S/X]G = [S/X]G_1 \,|[\Gamma]|\, [S/X]G_2$.
Let $[R/X]G_1 \,|[\Gamma]|\, [R/X]G_2 \xrightarrow{\alpha} R'$. Depending on whether $\underline{\text{name}}(\alpha) \in \Gamma$ or not, this action comes from the synchronisation of both processes or from the action of one of them only. We shall consider the case of the synchronisation only, which is more complete. So, we can write:
$[R/X]G_1 \,|[\Gamma]|\, [R/X]G_2 \xrightarrow{\alpha} R_{1'} \,|[\Gamma]|\, R_{2'}$ with $\underline{\text{name}}(\alpha) \in \Gamma$.
Then, $[R/X]G_1 \xrightarrow{\alpha} R_{1'}$ and $[R/X]G_2 \xrightarrow{\alpha} R_{2'}$.
Hence, by induction, we have $[S/X]G_1 \xrightarrow{\alpha} S_{1''} \sim S_{1'}$ with $\langle R_{1'}, S_{1'} \rangle \in \mathscr{R}$
and $[S/X]G_2 \xrightarrow{\alpha} S_{2''} \sim S_{2'}$ with $\langle R_{2'}, S_{2'} \rangle \in \mathscr{R}$.
Finally, by rule (PC3), we have $[S/X]G_1 \,|[\Gamma]|\, [S/X]G_2 \xrightarrow{\alpha} S_{1''} \,|[\Gamma]|\, S_{2''} \sim S_{1'} \,|[\Gamma]|\, S_{2'}$.
It remains to show that $\langle R_{1'} \,|[\Gamma]|\, R_{2'}\, S_{1'} \,|[\Gamma]|\, S_{2'} \rangle \in \mathscr{R}$.
But $\langle R_{i'}, S_{i'} \rangle \in \mathscr{R} (i = 1, 2)$, so for some H_i, $R_{i'} = [R/X]H_i$ and $S_{i'} = [S/X]H_i$.
So, we have $\langle [R/X]H_{1'} \,|[\Gamma]|\, [R/X]H_{2'}, [S/X]H_{1'} \,|[\Gamma]|\, [S/X]H_{2'} \rangle \in \mathscr{R}$.
The proof is similar for the time transitions, which also require a synchronisation.

Case 7: $G = \mathbf{Achoice}(d_1)\ x{:}s \Box G_1$.
Then, $[R/X]G = \mathbf{Achoice}(d_1)\ x{:}s \Box [R/X]G_1$
and $[S/X]G = \mathbf{Achoice}(d_1)\ x{:}s \Box [S/X]G_1$.
Let $\mathbf{Achoice}(d_1)\ x{:}s \Box [R/X]G_1 \xrightarrow{\alpha} R'$, with $d_1 > 0$. (The proof is simpler with $d_1 = 0$).
Then for some tx, $[tx/x][R/X]G_1 \xrightarrow{d_1} R_{1'}$ and $R_{1'} \xrightarrow{\alpha} R'$.
Hence, by induction, we have $[tx/x][S/X]G_1 \xrightarrow{d_1} S_{1''} \sim S_{1'}$ with $\langle R_{1'}, S_{1'} \rangle \in \mathscr{R}$.
As $\langle R_{1'}, S_{1'} \rangle \in \mathscr{R}$ by induction, we also have: $S_{1'} \xrightarrow{\alpha} S''' \sim S'$ with $\langle R', S' \rangle \in \mathscr{R}$
and $S_{1''} \sim S_{1'}$ implies $S_{1''} \xrightarrow{\alpha} S'''' \sim S'$.
Finally, by rule (GC2), we have $\mathbf{Achoice}(d_1)\ x{:}s \Box [S/X]G_1 \xrightarrow{\alpha} S'''' \sim S'$.
Let $\mathbf{Achoice}(d_1)\ x{:}s \Box [R/X]G_1 \xrightarrow{d} \mathbf{Achoice}(d + d_1)\ x{:}s \Box [R/X]G_1$.

Then $\forall tx \in \Theta(s) \bullet [tx/x][R/X]G_1 \xrightarrow{d+d_1}$.
Hence, by induction, we have $\forall tx \in \Theta(s) \bullet [tx/x][S/X]G_1 \xrightarrow{d+d_1}$.
Finally, by rule (GC3), we have:
Achoice$(d_1)\ x{:}s \Box [S/X]G_1 \xrightarrow{d} $ **Achoice**$(d+d_1)\ x{:}s \Box [S/X]G_1$.
And clearly, $\langle$**Achoice**$(d+d_1)\ x{:}s \Box [R/X]G_1$, **Achoice**$(d+d_1)\ x{:}s \Box [S/X]G_1 \rangle \in \mathscr{R}$.

Case 8: $G =$ **inf** $|||\ G_1$.
Then, $[R/X]G =$ **inf** $|||\ [R/X]G_1$
and $[S/X]G =$ **inf** $|||\ [S/X]G_1$.
Let **inf** $|||\ [R/X]G_1 \xrightarrow{\alpha} R' |||$ **inf** $|||\ [R/X]G_1$.
Then, $[R/X]G_1 \xrightarrow{\alpha} R'$.
Hence, by induction: $[S/X]G_1 \xrightarrow{\alpha} S'' \sim S'$ with $\langle R', S' \rangle \in \mathscr{R}$.
So, by rule (IP1): **inf** $|||\ [S/X]G_1 \xrightarrow{\alpha} S'' |||$ **inf** $|||\ [S/X]G_1 \sim S' |||$ **inf** $|||\ [S/X]G_1$.
It remains to show that $\langle R' ||| $ **inf** $|||\ [R/X]G_1, S' |||$ **inf** $|||\ [S/X]G_1 \rangle \in \mathscr{R}$.
But $\langle R', S' \rangle \in \mathscr{R}$, so for some H, $R' = [R/X]H$ and $S' = [S/X]H$,
and we have $\langle [R/X]H |||$ **inf** $|||\ [R/X]G_1, [S/X]H |||$ **inf** $|||\ [S/X]G_1 \rangle \in \mathscr{R}$.
The proof is simpler for the time transitions.

Case 9: $G =$ **hide** g **in** G_1.
Then, $[R/X]G =$ **hide** g **in** $[R/X]G_1$
and $[S/X]G =$ **hide** g **in** $[S/X]G_1$
Let **hide** g **in** $[R/X]G_1 \xrightarrow{\mathrm{i}}$ **hide** g **in** R'. Either this transition is derived from (H1) or (H2).
Consider that it is derived from (H2) (the demonstration is similar with (H1)).
Then, $[R/X]G_1 \xrightarrow{\alpha} R'$ with $\underline{\text{name}}(\alpha) = g$.
Hence, by induction: $[S/X]G_1 \xrightarrow{\alpha} S'' \sim S'$ with $\langle R', S' \rangle \in \mathscr{R}$,
and by rule (H2), **hide** g **in** $[S/X]G_1 \xrightarrow{\mathrm{i}} S'' \sim S'$.
If **hide** g **in** $[R/X]G_1 \xrightarrow{d}$ **hide** g **in** R' then $[R/X]G_1 \xrightarrow{d} R'$ and $\forall d_1 < d \bullet R_{(d_1)} \xrightarrow{g}$.[1]
Hence, by induction we have: $[S/X]G_1 \xrightarrow{d} S'' \sim S'$ with $\langle R', S' \rangle \in \mathscr{R}$,
and also $\forall d_1 \in (0, d) \bullet [S/X]G_1 \xrightarrow{d_1} S'_{(d_1)} \sim S_{(d_1)}$ with $\langle R_{(d_1)}, S_{(d_1)} \rangle \in \mathscr{R}$.
We still have to prove that $\forall d_1 < d \bullet S_{(d_1)} \xrightarrow{g}$. This is immediate however by lemma A4.1 (remember that by time additivity, $S'_{(d_1)} \xrightarrow{d-d_1}$ and $R'_{(d_1)} \xrightarrow{d-d_1}$).
So, by rule (H3), **hide** g **in** $[S/X]G_1 \xrightarrow{d}$ **hide** g **in** $S'' \sim$ **hide** g **in** S'.
And clearly, $\langle$**hide** g **in** R', **hide** g **in** $S' \rangle \in \mathscr{R}$.

Case 10: $G = G_1 >>$ **accept** $x_1{:}s_1, \ldots x_n{:}s_n$ **in** G_2.
Similar to case 9.

Case 11: $G = [SP] \to G_1$.
Then, $[R/X]G = [SP] \to [R/X]G_1$
and $[S/X]G = [SP] \to [S/X]G_1$.
If $\vdash SP$, the demonstration is similar to case 4 with $d_1 = 0$.
If $\neg \vdash SP$, $([SP] \to [R/X]G_1) \sim$ **stop** $\sim ([SP] \to [S/X]G_1)$ and the proof is immediate.

Case 12: $G =$ **let** $x_1 = tx_1, \ldots x_n = tx_n$ **in** G_1.
Similar to case 4 with $d_1 = 0$.

Case 13: $G = R_1[g_1, \ldots g_n](tx_1, \ldots tx_n)$ with $R_1[h_1, \ldots h_n](x_1, \ldots x_n) \coloneqq Q_1$.

[1] $R_{(d)} = ([R/X]G_1)_{(d)}$.

$R_1[g_1, \ldots g_n])tx_1, \ldots tx_n)$ is a constant and does not contain variable, so it is not influenced by the instantiation with R or S and the proof is immediate. □

Annex A5: About the isomorphism of the (ET-LOTOS, $\sim_E$) and (LOTOS, $\sim$) theories

We have already discussed the question of whether ET-LOTOS is a conservative extension of LOTOS in Section 7.5. It remains to prove the isomorphism of the (ET-LOTOS, $\sim_E$) and (LOTOS, $\sim$) theories for guarded LOTOS processes, i.e. P and Q being guarded LOTOS processes: $P \sim_E Q$ iff $P \sim Q$. This is the purpose of this annex.

A5.1. A formal definition of the notion of "guarded LOTOS process"

First of all, we need to define what we mean by a guarded (ET-)LOTOS specification.

A classical definition is the following: "P **where** $X_1 := P_1, \ldots X_n := P_n$" is a guarded specification if, by recursively substituting a finite number of times the expressions P_i's for the process identifiers X_i's occurring in P and in the P_i's themselves, it is possible to obtain an expanded specification "Q **where** $X_1 := Q_1, \ldots X_n := Q_n$" where Q and the Q_i's are guarded expressions, i.e. if all instantiations of X_i's in Q_i's are preceded by at least an action.

In our case, however, this definition is not sufficient. A non-zero delay or a false guard should also be considered as a sufficient prefixing. For example: P **where** $P := [false] \rightarrow P \sim$ **stop**. The problem is that the guard or the delay operator can contain variables[1] instantiated by operators placed higher in the syntactic structure. For example: **choice** $x : time \square [SP(x)] \rightarrow Q$. Apart from the action prefix, there are two operators in ET-LOTOS which can instantiate variables: **choice** and **let**.

Then, we propose a new definition for guarded specifications, better suited to ET-LOTOS. It is based on "guarded syntactic trees" (gs-trees for short). Given the specification "P **where** $X_1 := P_1, \ldots X_n := P_n$", one can represent P by a gs-tree. Each node of the gs-tree is labelled by a process. The root is labelled by P.

Some nodes only have one branch. This happens when the structure of the label process is: $\Delta^0 P$, **inf** $|||\ P$, **hide** Γ **in** P, $P >>$ **accept** $x_1 : s_1, \ldots x_n : s_n$ **in** P', $[SP] \rightarrow P$ with $\vdash SP$. In all these cases, the branch goes to the gs-tree of P.

Some nodes have two branches. This happens when the structure of the label process is: $P' \square P''$, $P' \|[\Gamma]\| P''$, $P'\ [> P''$. In all these cases, one branch goes to the gs-tree of P' and the other to the gs-tree of P''.

Some nodes are leaves: $go_1 \ldots o_n @ t[SP]$; P, $\mathbf{i}@t$; P, $\Delta^d P$ with $d > 0$, **exit**$\{d\}$, **stop**, $[SP] \rightarrow P$ with $\neg \vdash SP$.

Finally, some nodes require a special treatment: **Achoice**(d) $x_1 : s_1, \ldots x_n : s_n \square P$ has one branch to every possible $[tx_1/x_1, \ldots tx_n/x_n]P_{(d)}$. **let** $x_1 = tx_1, \ldots x_n = tx_n$ **in** P has one branch to the gs-tree of $[tx_1/x_1, \ldots tx_n/x_n]P$. And any X_i has one branch, which goes to the gs-tree of P_i.

[1] The delay value for the delay operator.

Then, we can give the following definition (where a path in a tree is a sequence of consecutive branches):

Definition A5.1
A specification "P **where** $X_1 := P_1, \ldots X_n := P_n$" is initially guarded iff the gs-tree of P contains no infinite path.[2]

Note that definition A5.1 does not meet the intuition of the "basic" definition given previously. For example a; P **where** $P := P$ is initially guarded but not guarded. This explains the name "initially" guarded: definition A5.1 ensures that it is possible to prefix any process name in P, but not that there is no risk for P to become unguarded after some transitions. The question, then, is how to characterize guarded specifications. One could extend the condition on P to all the X_i's. Intuitively, this would correspond to the "basic" definition. We consider however that this would be too restrictive. Take the following example: $[false] \rightarrow P$ **where** $P := P$. From a "basic" point of view, this specification should be considered as unguarded, although the definition of P clearly has no influence. The definition we propose is more comprehensive: a guarded specification is an initially guarded specification that remains so after any possible sequence of transitions:

Definition A5.2
"P **where** $X_1 := P_1, \ldots X_n := P_n$" is guarded iff:
(i) it is initially guarded
(ii) $\forall v \in A \cup D_{0\infty} \bullet P \xrightarrow{v} P'$ implies that "P' **where** $X_1 := P_1, \ldots X_n := P_n$" is guarded.
Clearly, any specification verifying the "basic" definition also verifies this one, for if an X_i appears unguarded in P', it is sufficient to replace it by the corresponding P_i, and so on.

A5.2. Proof of the isomorphism

A5.2.1. Some useful lemmas

We start by demonstrating some lemmas which will be useful for our proof.

Lemma A5.1
Let P be an ET-LOTOS process. If $P \sim_E P_{(d)}$ for some $d > 0$, then $\forall d_1 \in D_{0\infty} \bullet P \xrightarrow{d_1}$

Proof

By the time additivity property, the lemma is verified for any $d_1 \in (0, d)$.
As $P \sim_E P_{(d)}$: $P \xrightarrow{d} P_{(d)}$ implies $P_{(d)} \xrightarrow{d} P_{(d)(d)}$.
Hence, by the time additivity property, $P \xrightarrow{d+d} P_{(d)(d)}$. So, the lemma is verified for every $d_1 \in (0, d+d)$.
Furthermore, $P_{(d)} \sim_E P_{(d)(d)}$ and then $P \sim_E P_{(d)(d)}$. And so on.

Lemma A5.2: Let P be a LOTOS process. Then $\forall d \in D_{0\infty} \bullet P \xrightarrow{d} P_{(d)} \Rightarrow P \sim_E P_{(d)}$.
Note that with lemma A5.1, lemma A5.2 implies that a LOTOS process either cannot idle at all or can idle for any time value. Furthermore, when it idles, it always remains strongly bisimilar to itself.

[2] This does not mean that there is a finite upper bound to the lengths of the paths. Due to the choice over values, there can be an infinite number of paths, and then, no maximum to their lengths.

Proof

The proof can be done by induction upon the level of the transition $P \xrightarrow{d} P_{(d)}$. We argue by case on the syntactic structure of P.

Case 1: $P = \mathbf{stop}$ $\quad P = go_1 \ldots o_n[SP]; Q$
$P = \mathbf{exit}(e_1 \ldots e_n)$ $\quad P = \mathbf{i}; Q$

In all these cases, for every possible values of d, we have: $P_{(d)} = P$ and the proof is immediate. (For $P = \mathbf{i}; Q$ in fact, P cannot idle).

Case 2: $P = Q \Box R$
$P_{(d)} = Q_{(d)} \Box R_{(d)}$ by rule (Ch2).
The transitions $Q \xrightarrow{d} Q_{(d)}$ and $R \xrightarrow{d} R_{(d)}$ being of lower level than $P \xrightarrow{d} P_{(d)}$, we find by induction that $Q \sim_{\mathrm{E}} Q_{(d)}$ and $R \sim_{\mathrm{E}} R_{(d)}$. Strong bisimulation being a congruence, the result is immediate.

Case 3: $P = Q \,|[\Gamma]|\, R$ $\quad P = [SP] \rightarrow Q$
$P = Q \,[> R$ $\quad P = \mathbf{let}\ x_1 = tx_1, \ldots x_n = tx_n\ \mathbf{in}\ Q$
$P = \mathbf{hide}\ g\ \mathbf{in}\ Q$ $\quad P = R[g_1, \ldots g_n](tx_1, \ldots tx_m)$ with $R[h_1, \ldots h_n](x_1, \ldots x_m) := Q$
$P = Q >> \mathbf{accept}\ x_1 : s_1 \ldots x_n : s_n\ \mathbf{in}\ R$

The proof in all these cases is similar to the one in case 2.

Case 4: $P = \mathbf{choice}\ x : s \Box Q$
$P_{(d)} = \mathbf{Achoice}(d)\ x : s \Box Q$ by rule (GC3).
Let us begin with the derivation of some useful facts:
$P \xrightarrow{d} P_{(d)}$ implies $\forall tx \in \Theta(s) \bullet [tx/x]Q \xrightarrow{d} [tx/x]Q_{(d)}$.
All these transitions being of lower level than $P \xrightarrow{d} P_{(d)}$, we find by induction that:
(1) $\forall tx \in \Theta(s) \bullet [tx/x]Q \sim_{\mathrm{E}} [tx/x]Q_{(d)}$.
Hence, by lemma A5.1:
(2) $\forall tx \in \Theta(s) \forall\, d_1 \in D_{0\infty} \bullet [tx/x]Q \xrightarrow{d_1} [tx/x]Q_{(d_1)}$.
Furthermore, as $[tx/x]Q_{(d)(d_1)} = [tx/x]Q_{(d+d_1)}$ by time additivity, we deduce from (1) and (2):
(3) $\forall tx \in \Theta(s) \forall\, d_1 \in D_{\infty} \bullet [tx/x]Q_{(d_1)} \sim_{\mathrm{E}} [tx/x]Q_{(d+d_1)}$.
With $\mathbf{Achoice}(d)\ x : s \Box Q$ however, the passing of time does not affect Q but the parameter d. Due to this situation, we cannot apply the method of case 2.
Remember that what we try to prove is: $P \sim_{\mathrm{E}} P_{(d)}$ assuming that $P' \sim_{\mathrm{E}} P'_{(d_1)}$ for every couple (P', d_1) such that $P' \xrightarrow{d_1} P'_{(d_1)}$ is of lower level than $P \xrightarrow{d} P_{(d)}$. Another common way to prove that two processes are strongly bisimilar is to find a strong bisimulation relation containing them.
Given $P = \mathbf{choice}\ x : s \Box Q$ and a time value d, consider the following relation (which actually contains $\langle \mathbf{choice}\ x : s \Box Q, \mathbf{Achoice}(d)\ x : s \Box Q \rangle$):
$\mathscr{R} = \mathscr{R}1 \cup \mathscr{R}2$,
with $\mathscr{R}1 = \{\langle R, S \rangle \mid R \sim_{\mathrm{E}} S\}$
and $\mathscr{R}2 = \{\langle \mathbf{Achoice}(d_1)\ x : s \Box Q, \mathbf{Achoice}(d + d_1)\ x : s \Box Q \mid d_1 \in D_{\infty}\}$.
To prove that $\mathscr{R}$ is a strong bisimulation, we have to show that every couple in $\mathscr{R}2$ verifies the transfer property (for the couples in $\mathscr{R}1$ this is immediate).
So, for time transitions:
By rule (GC3), we can derive from (2):
$\forall\, d_1 \in D_{\infty} \forall\, d_2 \in D_{0\infty} \bullet \mathbf{Achoice}(d_1)\ x : s \Box Q \xrightarrow{d_2} \mathbf{Achoice}(d_1 + d_2)\ x : s \Box Q$
and $\mathbf{Achoice}(d + d_1)\ x : s \Box Q \xrightarrow{d_2} \mathbf{Achoice}(d + d_1 + d_2)\ x : s \Box Q$.

In other words, as $\langle$**Achoice**$(d+d_2)\ x{:}s\Box Q$, **Achoice**$(d+d_1+d_2)\ x{:}s\Box Q > \in \mathscr{R}$, the transfer relation for the time transitions is verified for any pair in $\mathscr{R}$.
And for action transitions:
By rule (GC2) (or (GC1)):
Achoice$(d_1)\ x{:}s\Box Q \xrightarrow{\alpha} Q' \wedge [tx/x]Q_{(d_1)} \xrightarrow{\alpha} Q'$ for some tx.
Then, by (3): $[tx/x]Q_{(d+d_1)} \xrightarrow{\alpha} Q''$ with $Q' \sim_{\mathrm{E}} Q''$ (which implies $\langle Q', Q''\rangle \in \mathscr{R}$).
Hence, by (GC2), **Achoice**$(d+d_1)\ x{:}s\Box Q \xrightarrow{\alpha} Q''$, and the transfer relation is verified for action transitions (the same proof can be done by taking **Achoice**$(d+d_1)\ x{:}s\Box Q \xrightarrow{\alpha} Q'$ as starting point).

Lemma A5.3: Let P be a guarded LOTOS process. Then: $(\exists d \in D_{0\infty} \bullet P \xrightarrow{d}) \Leftrightarrow P \xrightarrow{\mathbf{i}}$.

Proof

$\bullet\ P \xrightarrow{d} \Rightarrow P \xrightarrow{\mathbf{i}}$.
The proof can be done by induction upon the level of $P \xrightarrow{d} P_{(d)}$. We argue by case on the structure of P.

Case 1: $P =$ **stop** $\quad P = go_1 \ldots o_n[SP]; Q$
$P =$ **exit**$(e_1 \ldots e_n)$ $\quad P = [SP] \rightarrow Q$ with $\vdash \neg\, SP$

In all these cases, P cannot perform **i** and the proof is straightforward.

Case 2: $P = \mathbf{i}; Q$
In this case, P cannot idle and the proof is straightforward.

Case 3: $P = Q \Box R$
By rule (Ch2), $P \xrightarrow{d}$ implies that $Q \xrightarrow{d}$ and $R \xrightarrow{d}$.
So, by induction, $Q \xrightarrow{\mathbf{i}}$ and $R \xrightarrow{\mathbf{i}}$.
Hence, $P \xrightarrow{\mathbf{i}}$.

Case 4: $P = Q\ |[\Gamma]|\ R$ $\quad P =$ **let** $x_1 = tx_1, \ldots x_n = tx_n$ **in** Q
$P = Q\ [> R$ $\quad P = [SP] \rightarrow Q$ with $\vdash SP$
$P =$ **choice** $x{:}s\Box Q$ $\quad P = R[g_1, \ldots g_n](tx_1, \ldots tx_m)$ with
$R[h_1, \ldots h_n](x_1, \ldots x_m) := Q$

The proof in all these cases is similar to the one in case 3.

Case 5: $P =$ **hide** g **in** Q
Two rules allow P to perform an **i**: (H1) and (H2). However:
By rule (H3), $P \xrightarrow{d}$ implies that $Q \xrightarrow{d}$.
So, by induction, $Q \xrightarrow{\mathbf{i}}$.
Hence, $P \xrightarrow{\mathbf{i}}$ by rule (H1).
$P \xrightarrow{d}$ implies also that $Q \xrightarrow{g}$.
Hence, $P \xrightarrow{\mathbf{i}}$ by rule (H2).

Case 6: $P = Q >>$ **accept** $x_1{:}s_1 \ldots x_n{:}s_n$ **in** R
This case is similar to the previous one.

$\bullet\ P \xrightarrow{\mathbf{i}} \Rightarrow (\forall d \in D_{0\infty} \bullet P \xrightarrow{d})$.
Remember that for guarded LOTOS processes, we have:
$(\forall d \in D_{0\infty} \bullet P \xrightarrow{d}) \Leftrightarrow (\exists d \in D_{0\infty} \bullet P \xrightarrow{d})$.
We would like to make our proof by induction, as usual. However, the induction cannot be based on the level of $P \xrightarrow{\mathbf{i}}$, that does not exist by definition. A solution is to take benefit of the fact that P is guarded. We define a partial order on guarded LOTOS processes: $P \propto Q$ iff p is the label of a node (except the root) in

the gs-tree of Q. This partial order is non-reflexive: if $P \propto P$, there would be an infinite path in the gs-tree of P. It is antisymmetric: in fact we cannot have: $P \propto Q$ and $Q \propto P$, otherwise there would be an infinite path in the gs-trees of P and Q. And it is transitive: $P \propto Q$ and $Q \propto R$ implies $P \propto R$: P appears in the subtree of R whose root is labelled by Q. Furthermore, it is not possible to have an infinite decreasing chain $(P_1, P_2, \ldots, P_n, \ldots)$ such that $P_{n+1} \propto P_n$: this chain would constitute an infinite path in the gs-tree of P_1. So, $\propto$ is valid for a proof by induction.
We argue by case on the syntactic structure of P.

Case 1: $P = \mathbf{stop}$ $\quad P = \mathbf{exit}(e_1 \ldots e_n)$
$P = go_1 \ldots o_n[SP]; Q$ $\quad P = [SP] \rightarrow Q$ with $\neg\vdash SP$

In all these cases, P can idle without any restriction.

Case 2: $P = \mathbf{i}; Q$
In this case, P can perform **i**.

Case 3: $P = Q \Box R$
By rules (Ch1) and (Ch1′), $P \overset{\mathbf{i}}{\not\rightarrow}$ implies that $Q \overset{\mathbf{i}}{\not\rightarrow}$ and $R \overset{\mathbf{i}}{\not\rightarrow}$. By definition of the gs-tree of P, we have: $Q \propto P$ and $R \propto P$. So, by induction, $\forall d \in D_{0\infty} \bullet Q \overset{d}{\rightarrow}$ and $\forall d \in D_{0\infty} \bullet R \overset{d}{\rightarrow}$. Hence, by rule (CH2) $\forall d \in D_{0\infty} \bullet P \overset{d}{\rightarrow}$.

Case 4:
$P = Q \,[[\Gamma]]\, R$ $\quad P = Q >> \mathbf{accept}\ x_1 : s_1 \ldots x_n : s_n\ \mathbf{in}\ R$
$P = Q \,[> R$ $\quad P = [SP] \rightarrow Q$ with $\vdash SP$
$P = \mathbf{choice}\ x : s \Box Q$ $\quad P = \mathbf{let}\ x_1 = tx_1, \ldots x_n = tx_n\ \mathbf{in}\ Q$
$P = \mathbf{hide}\ g\ \mathbf{in}\ Q$ $\quad P = R[g_1, \ldots g_n](tx_1, \ldots tx_m)$ with $R[h_1, \ldots h_n](x_1, \ldots x_m) := Q$

The proof in all these cases is similar to the one in case 3.

Lemma A5.4: Let P be a LOTOS process. Then, $\forall\ \alpha \in A \bullet P \overset{\alpha}{\rightarrow} P' \Leftrightarrow P \overset{\alpha}{\rightarrow}_{\mathrm{E}} P'$.
In the remainder of this proof, we denote by $\overset{\alpha}{\rightarrow}_{\mathrm{E}}$ the transitions derived from the ET-LOTOS theory to distinguish them from the LOTOS ones.

The proof of lemma A5.4 is straightforward: in what concerns the actions, both theories share the same rules as far as they are applied on LOTOS terms.

A5.2.2. Proof of the isomorphism
Let us prove now the isomorphism of (ET-LOTOS, $\sim_{\mathrm{E}}$) and (LOTOS, $\sim$) for the guarded LOTOS processes.
Let P and Q be two guarded LOTOS processes. Then: $P \sim Q \Leftrightarrow P \sim_{\mathrm{E}} Q$.

• $P \sim Q$ implies $P \sim_{\mathrm{E}} Q$ for guarded LOTOS processes.
It is sufficient to show that the following relation is a strong bisimulation in ET-LOTOS:
$\mathscr{R} = \{\langle P, Q\rangle \mid P, Q$ are ET-LOTOS processes
and $P \sim_{\mathrm{E}} P_1 \sim Q_1 \sim_{\mathrm{E}} Q$ for some guarded LOTOS processes P_1 and $Q_1\}$,
because if P and Q are guarded LOTOS processes, we have:
$P \sim Q$ implies $P \sim_{\mathrm{E}} P \sim Q \sim_{\mathrm{E}} Q$ implies $\langle P, Q\rangle \in \mathscr{R}$.
Let $\langle P, Q\rangle \in \mathscr{R}$, with $P \sim_{\mathrm{E}} P_1 \sim Q_1 \sim_{\mathrm{E}} Q$, P_1 and Q_1 being guarded LOTOS processes.
Then, for the action transitions:
$P \overset{\alpha}{\rightarrow}_{\mathrm{E}} P'$ implies $P_1 \overset{\alpha}{\rightarrow}_{\mathrm{E}} P_{1'}$, for some $P_{1'}$, with $P' \sim_{\mathrm{E}} P_{1'}$.

According to lemma A5.4, $P_1 \stackrel{g}{\rightarrow}_E P_{1'}$ implies $P_1 \stackrel{g}{\rightarrow} P_{1'}$ and $P_{1'}$ is a guarded LOTOS process.[1]
So, $Q_1 \stackrel{g}{\rightarrow} Q_{1'}$ for some $Q_{1'}$, with $P_{1'} \sim Q_{1'}$ and $Q_{1'}$ is a guarded LOTOS process.
Also, according to lemma A5.4, $Q_1 \stackrel{g}{\rightarrow} Q_{1'}$ implies $Q_1 \stackrel{g}{\rightarrow}_E Q_{1'}$.
Finally, $Q_1 \stackrel{g}{\rightarrow}_E Q_{1'}$ implies Q $\stackrel{g}{\rightarrow}_E Q'$ for some Q', with $Q' \sim_E Q_{1'}$.
So, we have $P' \sim_E P_{1'}$, $Q' \sim_E Q_{1'}$, $P_{1'} \sim Q_{1'}$, $P_{1'}$ and $Q_{1'}$ being guarded LOTOS processes. Hence, $\langle P', Q' \rangle \in \mathscr{R}$.
And for the time transitions:
According to lemma A5.3, $P \stackrel{d}{\rightarrow}_E P_{(d)}$ implies $P \stackrel{i}{\not\rightarrow}_E$. Hence, $P_1 \stackrel{i}{\not\rightarrow}_E$.
According to lemma A5.4, $P_1 \stackrel{i}{\not\rightarrow}_E$ implies $P_1 \stackrel{i}{\not\rightarrow}$. Hence, $Q_1 \stackrel{i}{\not\rightarrow}$
According to lemma A5.4, $Q_1 \stackrel{i}{\not\rightarrow}$ implies $Q_1 \not\rightarrow$ E. Hence, $Q \stackrel{i}{\not\rightarrow}_E$.
According to lemma A5.3, $Q \stackrel{i}{\not\rightarrow}_E$ implies $Q \stackrel{d}{\rightarrow}_E Q_{(d)}$.
Applying lemma A5.1, we find $P_{(d)} \sim_E P \sim_E P_1$, $Q_{(d)} \sim_E Q \sim_E Q_1$. As P_1 and Q_1 are guarded LOTOS processes with $P_1 \sim Q_1$, then $\langle P_{(d)}, Q_{(d)} \rangle \in \mathscr{R}$.

• $P \sim_E Q$ implies $P \sim Q$ for guarded LOTOS processes.
It is sufficient to show that the following relation is a strong bisimulation in LOTOS:
$\mathscr{R} = \{\langle P, Q \rangle \mid P$ and Q are guarded LOTOS processes and $P \sim_E Q\}$,
For then, P and Q being guarded LOTOS processes, we find: $P \sim_E Q$ implies $\langle P, Q \rangle \in \mathscr{R}$.
Let $\langle P, Q \rangle \in \mathscr{R}$, with $P \sim_E Q$, P and Q being guarded LOTOS processes.
According to lemma A5.4, $P \stackrel{g}{\rightarrow} P'$ implies $P \stackrel{g}{\rightarrow} \sim_E P'$ and P' is a guarded LOTOS process.
So, $Q \stackrel{g}{\rightarrow}_E Q'$, with $P' \sim_E Q'$ and Q' is a guarded LOTOS process.
Also, according to lemma A5.4, $Q \stackrel{g}{\rightarrow}_E Q'$ implies $Q \stackrel{g}{\rightarrow} Q'$.
We have $P' \sim_E Q'$. Hence, $\langle P', Q' \rangle \in \mathscr{R}$. □

[1] Remember that by definition, guarded LOTOS processes remain so after any transition.