On an effective hierarchy of communicating processes: separation principle and testing

John Mullins

Lab. d' Automatique de Nantes, Ecole Centrale de Nantes, 1, rue de la Noe, Nantes, France

Communicated by M. Nivat

Abstract

This paper is concerned with an effective hierarchy of guaranteed processes. A process satisfies a guarantee requirement when that requirement holds at some forthcoming point of every computation. $\Sigma^0_1$ being the class of $\omega$-languages accepted by the base of this process hierarchy, this class of processes, namely $\mathcal{C}_1$, is a class of universal machines. Translated in these terms, our results throw light on, first, the increased computational power of the machine resulting from communication between any finite number of such machines and, second, that the hierarchy whose next degree represents $\omega$-languages accepted by machines resulting from communication between any finite number of machines in the current degree, is infinite. Interaction product acts as jump operator.

We naturally start with some effective classes of guarantee properties proven to form a hierarchy and first prove that they have not the separation property. A class $\mathcal{C}$ of $\omega$-languages has the separation property if, for any pair $(U, V)$ in $\mathcal{C}$, there exists an $\omega$-language $W \in \mathcal{C}$ such that $U \cap W = \emptyset \Leftrightarrow V \cap W \neq \emptyset$. We then induce non-separability under testing from the above language theoretic non-separability result. Two processes are said to be separable if they can be separated from each other by means of another test process from the same class.

It finally turns out that some processes, having different visible behaviours, are test equivalent. We close the paper on logical complexity issues of the testing problem when test criteria and guarantee constraints range over the arithmetical hierarchy.

1. Introduction

The well-known Church thesis claims that any universal model for sequential computation has the same computational power as Turing machines. There are some reasons why no such thesis has been developed for concurrent and communicating systems. One of these reasons is the wide variety of equivalences defined on

---

1 This work has been supported by an NSERC grant (Canadian Government).

2 On leave from: Department of Computer Science, Ottawa, University of Ottawa, Canada K1N 6N5. E-mail: mullins@csi.uottawa.ca.
communicating systems mainly due to the wide number of interpretations which may be defined for atomic actions: different equivalences inducing different quotient models.

Nevertheless, equivalences and models may be translated in the uniform framework of recursive automata and compared by means of their logical complexity. Due to the high degree of inefficiency of concurrency models, this complexity has to be graded in arithmetical and analytical hierarchies.

Testing equivalences are among these equivalences. In [2], a separation principle is defined for classes of $\omega$-languages which is concerned with the general notion of test. Therefore, the motivation for such a separation principle is neither topological nor recursive theoretic, where the use of another, well-known separation principle has been so fruitful (see for example [1]); neither is in despite the nature of the methods used, a language theoretic motivation; it is clearly a concurrent system theoretic one.

The starting point is based on the principle that, in the framework of "observational" semantics of a process algebra, an extensional definition of any process can be seen as an $\omega$-language: one of its infinite sequences of visible actions. Let $|P|$ denote the set of visible behaviours of $P$. Two processes $P$ and $Q$ in a given class will be separable if we can produce a test set through another process $T$ in the same class such that $|P| \cap |T| = \emptyset$ iff $|Q| \cap |T| \neq \emptyset$. It is proved in [2], based on the fact that $\Sigma$ has not the separation property, that there does not exist any of test which would permit to discriminate any pair of CCS processes having distinct observational behaviours. It seems apparently nonconsistent with the very well known existence of fully abstract models for CCS, full abstraction [5] being the property according to which denotations always discriminate distinct process behaviours in all possible contexts. Based on these separability results, several generalizations of testing have been proposed in [3] to overcome the problem.

The first question to arise is what are the separable classes of processes? This work answers the question for classes from a particular hierarchy of guarantee. A process satisfies a guarantee property if a phenomenon of a certain kind is guaranteed to be observable in finite time in any infinite sequence of its visible actions. For example, total correctness is a guarantee property. The phenomenon guaranteed to occur in that case is that the process will eventually reach a final state. Sets of infinite sequences satisfying some guarantee property can therefore be expressed in the form $U X^\omega$: $X$ being the set of actions and $U$, the set of finite behaviours terminating in a guaranteed state.

In a computational framework, it is natural to demand finite process behaviours to be "effectively" generated by a computational device. That is why we focus on the class of guarantee properties generated when $U$ ranges over the arithmetical hierarchy. It is known that, when $X$ is finite, we get in that way, an infinite hierarchy not bounded by any arithmetical class of $\omega$-languages, i.e. with arbitrary effective structural complexity [9].

Our first result is to prove that, even in the general case, i.e. even when $X$ is infinite, none of these classes can be separated (Theorem 3.2).
As the second step, we will define, a class of nondeterministic recursive automata whose for each of the above class, we will see that the trace sets of the former coincide exactly with the latter. This result appears to be independent from the finiteness of the set of actions as well as from the branching factor of automata.

As the next step, we will define an interaction mechanism between recursive automata "à la CCS". It appears that the product automaton jumps to the upper degree of the hierarchy. We then define a Hennessy-like notion of test and test equivalences [4] taking into account guarantee constraint and we will prove non-separability under testing for each class of automata (Theorem 6.2). This result is mainly based on the capability to remove second-order quantifiers we have to deal with (Lemma 5.1).

All previous considerations finally lead to a concise presentation of the logical complexity of the testing problem (Theorem 7.1). We prove that the problem is in \( \Pi^*_{\alpha+3} \) when test criteria and guarantee constraints are \( \Sigma^*_\alpha \) properties.

To close the introduction, let us mention some related results. In [10], the author investigates the logical complexity of the language containment problem for several classes of recursive automata on infinite words over a finite alphabet. It is shown in particular that, under the assumption of finite alphabet, solving the equation \( |P| = \emptyset \) is \( \Pi^*_{\alpha} \)-complete for \( |P|, |T| \in \Sigma^0_\alpha \). Ref. [3] is more closely related to the testing problem: it investigates logical complexity issues for several forms of finitary and infinitary testing equivalence on CCS. It is proved in particular that in the case of De Nicola and Hennessy's equivalence, the testing problem for CCS is in \( \Pi^*_{\beta} \). CCS's assumption of branching finiteness is therefore assumed.

An earlier version of this work has been presented at ICTCS'95 [7].

2. Background and preliminary definitions

Let \( X \) denotes an alphabet, and \( X^* \) and \( X^\omega \), the set of respectively finite words and infinite sequences on alphabet \( X \). The empty word is denoted by \( \varepsilon \). Given \( \alpha \in X^* \cup X^\omega \), throughout the paper \( \alpha[i] \) stands for the prefix of length \( i \) of \( \alpha \) while \( \alpha(i) \) stands for the \( i \)th letter of \( \alpha \) and \( |\alpha| \) for the length of \( \alpha \). Subsets of \( X^* \) and of \( X^\omega \) will be called languages and \( \omega \)-languages respectively. They will be denoted indistinctly using capital Roman letters (\( U, V, \ldots \)). Let \( u \alpha \) be the concatenation of \( u \in X^* \) and \( \alpha \in X^\omega \), it can be extended up to a product \( UV \) of a language \( U \) and an \( \omega \)-language \( V \) in the obvious way. We will denote \( uV \) as a shortcut for \( \{u\}V \). In the sequel, we will denote words using lower-case Roman letters (\( u, v, \ldots \)) and infinite sequences by lower-case Greek letters (\( \alpha, \beta, \ldots \)).

It is natural to see \( X^\omega \) as a topological space and more precisely as the denumerable product of \( X \) by itself with \( X \) equipped with the discrete topology (every letter of the alphabet \( X \) is an open set).

Using Tychonoff's theorem, it is easily seen that \( X^\omega \) has the compactness property if and only if \( X \) is finite. Indeed, discrete topology has this property only in this case.
The topology described above is known to be homeomorphic to Baire space. An $\omega$-language $W$ is an open set if and only if there is a language $U$ such that $W = \{ x : \exists i x[i] \in U \}$ and is a closed set if and only if there is a language $U$ such that $W = \{ x : \forall i x[i] \in U \}$. So the property to be open or closed for an $\omega$-language may be seen as a "limit" of some finitary language, i.e. as the result of a limit operator from the powerset $2^{X^*}$ to $2^{X^\omega}$ expressing exactly $\Sigma_n^0$ and $\Pi_n^0$ as "limits" of languages and appearing to be dual for each other: let $U \subseteq X^*$, then define

$$\lim_{\Sigma_i} U \overset{\text{def}}{=} \{ x \in X^\omega : \exists i x[i] \in U \},$$

$$\lim_{\Pi_i} U \overset{\text{def}}{=} \{ x \in X^\omega : \forall i x[i] \in U \}.$$

The generalization of the notions of open and closed sets provides the well-suited framework to study the structural properties of $\omega$-languages. Borel $\omega$-languages are the smallest family of $\omega$-languages closed under denumerable unions and intersections and containing the open sets. In terms of predicates defining these sets, it is the smallest class closed under projection and complementation. We will make use of this alternative and perhaps more unusual definition which can be found in [9]: a Borel $\omega$-language is an $\omega$-language of the form

$$W = \{ x \in X^\omega : \exists_{l_n} \exists_{l_0} \cdots \exists_{l_1} P(t_1, t_2, \ldots, t_{n-1}, x[t_n]) \},$$

where $\exists_{l_i}$ are arbitrary quantifiers and $P$ is a predicate of $n - 1$ integer variables and one variable on $X^*$ corresponding to a relation $M_P$ on $\omega^{n-1} \times X^*$ in the following way:

$$P(t_1, t_2, \ldots, t_{n-1}, x[t_n]) \text{ iff } (t_1, t_2, \ldots, t_{n-1}, x[t_n]) \in M_P.$$

The Borel hierarchy of $\omega$-languages is defined as follows: $W \in \Sigma_n^0$ if $\exists_{l_n} \equiv \exists_0$ and $W \in \Pi_n^0$ if $\exists_{l_n} \equiv \forall_0$. It turns out that $\Sigma_0^0 = \Pi_0^0 = \{ U^{X^\omega} : U \subseteq X^* \text{ finite} \}$, i.e. the family of clopen sets while $\Sigma_1^0$ and $\Pi_1^0$ correspond to open and closed sets respectively.

Arithmetical $\omega$-languages are obtained in the same way but adding the constraint on the predicate $P$ to be recursive, i.e. corresponding to a recursive relation $M_P$ on $\omega^{n-1} \times X^*$ as above. The arithmetical hierarchy of $\omega$-languages is defined in the same fashion: $W \in \Sigma_n^0$ if $\exists_{l_n} \equiv \exists_0$ and $W \in \Pi_n^0$ if $\exists_{l_n} \equiv \forall_0$. Notice the lightface font which distinguishes for example the class $\Sigma_1^0$ from the class $\Sigma_1^1$ of open sets. In this effective setting, $\Sigma_0^0 = \Pi_0^0 = \Sigma_0^0 = \Pi_0^0$ is the family of recursive sets of $X^\omega$ so $\Sigma_1^0$ is the class of recursively enumerable sets of $X^\omega$.

By induction one easily obtains that

$$\Sigma_n^0 \cup \Pi_n^0 (\text{resp. } \Sigma_n^0 \cup \Pi_n^0) \subset \Sigma_{n+1}^0 \cap \Pi_{n+1}^0 (\text{resp. } \Sigma_{n+1}^0 \cap \Pi_{n+1}^0),$$

(1)

$$U \in \Sigma_n^0 (\text{resp. } \Sigma_n^0) \text{ iff } \overline{U} \in \Pi_n^0 (\text{resp. } \Pi_n^0).$$

(2)

The relation between these two hierarchies in upper levels is stated as follows:

$$\Sigma_0^0 \subset \Sigma_n^0 \text{ if } n > 1.$$
We assume the reader to be familiar with the basic results on the arithmetical hierarchy and the Tarski-Kuratowski algorithm to rewrite formulas into normal forms [8] (Ch. 15). In particular, conversion rules to contract quantifiers and eliminate bounded quantifiers will be extensively used throughout the paper.

We will now introduce the hierarchy of guarantee properties that we intend to focus on. Bear in mind that a process satisfies a guarantee property if a phenomenon of a certain kind is guaranteed to be observable in finite time in any infinite sequence of its visible actions. Hence, they correspond to open sets in the Baire topology.

Let $W = \lim_{\omega} U$ be such a property. From a computational point of view, we could think of $U$ as the set of finite behaviours of the process, meaning that it should be “effectively” generated by some computational device. Hence, let us define now, effective languages.

We define ascetical languages as the class of languages $W$ of the form

$$W = \{ u \in X^* : \exists t_1, t_2, \ldots, t_n, u \}$$

with all the rest as above. $\Sigma_n^*$ and $\Pi_n^*$ denote classes of the arithmetical hierarchy of languages. Here again, we get a similar result to Eqs. (1) and (2). There are many interesting results about effective hierarchies generated by these limit operators [9] (Corollaries 4.3 and 5.5):

$$\lim_{\Sigma_1} \Sigma_1^* (\text{resp. } \lim_{\Pi_1^*} \Pi_1^*) = \Sigma_1^0 (\text{resp. } \Pi_1^0),$$

$$\lim_{\Sigma_1} \Sigma_n^* (\text{resp. } \lim_{\Pi_n^*} \Pi_n^*) \subset \Sigma_n^0 (\text{resp. } \Pi_n^0),$$

$$U \in \lim_{\Sigma_1} \Sigma_n^* \text{ iff } \bar{U} \in \lim_{\Pi_1^*} \Pi_n^*. $$

It is known that, when $X$ is finite, we get an hierarchy of “effectively generated” guarantee properties, not bounded by any arithmetical class of $\omega$-languages [9] (Section 6), i.e. properties with arbitrary effective structural complexity and more, each class of this hierarchy is closed under finite union and intersection and under projection [9] (Proposition 4.4).

3. The separability result for $\omega$-languages

The next definition has been widely motivated in the introduction.

Definition 3.1. A class $\mathcal{C}$ of $\omega$-languages will be said to be separable if for any pair $(U, V)$ in $\mathcal{C}$ there exists on $\omega$-language $W \in \mathcal{C}$ such that $U \cap W = \emptyset \iff V \cap W \neq \emptyset$.

A first result is immediate: for $n \geq 1$, $\Sigma_n^0 \cap \Pi_n^0$ and $\lim_{\Sigma_n^*} \Sigma_n^* \cap \lim_{\Pi_n^*} \Pi_n^*$ are separable. Indeed, these classes are closed under boolean operators. However, quite surprisingly, there is a majority of nonseparable $\omega$-language classes. In particular, it was proved in [2] that any $\omega$-language class obtained from $\omega$-Kleene closure of a
language class containing context-free languages and closed under finite union, intersection with regular language, concatenation product, Kleene closure, alphabetic morphisms and their inverses, are not separable. Context-free, recursive and recursively enumerable languages are such classes. Therefore, \( \omega \)-context-free languages as well as \( \Sigma_1^0 \) have not the separation property, being the \( \omega \)-Kleene closure of context-free and recursively enumerable languages, respectively. The next result extends these results, using a similar idea to one used in [2] (Theorem in II.4) to prove that \( \Sigma_1^1 \), a class properly containing arithmetical \( \omega \)-languages, is not separable.

**Theorem 3.2.** For \( n \geq 1 \), \( \lim_{\Sigma_1^0} \Sigma_n^* \) is not separable.

**Proof.** Let \( I \) be a recursive set of integers and \( x_0, x_1 \in X \). We define an \( \omega \)-sequence \( \beta \) such that

\[
\beta(i) = \begin{cases} 
  x_0 & \text{if } i \in I, \\
  x_1 & \text{otherwise}.
\end{cases}
\]

Clearly \( \{ \beta \} \in \Pi_1^0 \). Indeed, \( u \) is a prefix of \( \beta \) iff

\[
\forall x \in X \forall j \leq |u| (u(j) = x \rightarrow \beta(j) = x)
\]

and also because the condition \( \beta(j) = x \) is, by definition of \( \beta \), a \( \Pi_1^* \)-predicate. Hence, by the Tarski–Kuratowski algorithm, the predicate \("u is a prefix of \( \beta \)" is a \( \Pi_1^* \)-predicate.

Given \( U = X^\omega \). Then, \( U \in \lim_{\Sigma_1^0} \Sigma_n^* \) for any \( n \geq 1 \). We may now define \( V = U \setminus \{ \beta \} \). Clearly \( \{ \beta \} \notin \lim_{\Sigma_1^0} \Sigma_n^* \) because \( \lim_{\Sigma_1^0} \Sigma_n^* \) contains only infinite \( \omega \)-languages. By Eqs. (1)-(5), \( V \in \lim_{\Sigma_1^0} \Sigma_n^* \). If \( U \) and \( V \) were separable, there would exist \( W \) in \( \lim_{\Sigma_1^0} \Sigma_n^* \) such that \( U \cap W = \{ \beta \} \) and that is impossible: in such a case, \( \{ \beta \} \) would be in \( \lim_{\Sigma_1^0} \Sigma_n^* \) and this ends the proof. \( \Box \)

Using Eq. (3) we obtain immediately:

**Corollary 3.3.** \( \Sigma_1^0 \) is not separable.

So, Theorem 3.2 generalizes this and since one cannot conclude Theorem 3.2 from the non-separability of \( \Sigma_1^1 \), the above result refines [2] (Theorem in II.4).

4. From languages to traces and computations

We will now introduce some classes of non-deterministic automata and we will show that classes of accepted \( \omega \)-languages corresponds to \( \lim_{\Sigma_1^0} \Sigma_n^* \) classes. It will appear that the extension of our non-separability result to processes depends on this.

An automaton \( \mathcal{A} \) over an alphabet \( X \) is a 4-tuple \((Q, Q_0, Q_f, \rightarrow)\) where \( Q \) is a set of states, \( Q_0 \subseteq Q \) is the set of initial states \( Q_f \subseteq Q \) is the set of final states and \( \rightarrow \subseteq Q \times X \times Q \) is the transition relation. We will denote \( q \xrightarrow{x} q' \) for \((q, x, q') \in \rightarrow \).
An automaton $\mathcal{A}$ is said to be $n$-open provided $Q$ is recursive and $Q_0, \rightarrow, \epsilon$ and $Q_f \subseteq \Sigma_*^n$. We denote by $\mathcal{C}_n$ the $n$-open automata class. An automaton $\mathcal{A}$ is said to be deterministic if $Q_0$ and for every $x \in X$ and $q \in Q$, $\{q': q \xrightarrow{a} q'\}$ are singletons. An automaton $\mathcal{A}$ is said to be finitely, recursively or infinitely branching depending on if the set $\{q': q \xrightarrow{a} q'\}$ is finite, recursive or infinite for every $x \in X$ and $q \in Q$. An automaton is said to be complete if $\{q': q \xrightarrow{a} q'\} \neq \emptyset$ for every $x \in X$ and $q \in Q$. In this paper, unless it is otherwise indicated, all automata are assumed to be complete and arbitrarily branching.

Given a word $x \in X^*$ of length $n$, a sequence on $Q \times X \times Q$ is called a run of $\mathcal{A}$ over $x$ if

$$ q_0 \xrightarrow{a(0)} q_1 \xrightarrow{a(1)} \cdots \xrightarrow{a(n-1)} q_n $$

with $q_0 \in Q_0$. It is an accepting run if $q_n \in Q_f$. The set of words $L(\mathcal{A})$ with an accepting run of $\mathcal{A}$ is called the language accepted by $\mathcal{A}$.

For a sequence $x \in X^\omega$, a run of $\mathcal{A}$ over $x$ is an infinite sequence on $Q \times X$ such that

$$ q_0 \xrightarrow{a(0)} q_1 \xrightarrow{a(1)} \cdots $$

with $q_0 \in Q_0$. An accepting run for $x$ is then one such that $\exists i: q_i \in Q_f$, i.e. if it goes through $Q_f$ at least once. We extend some notations already used for languages, to computations: reserving lower-case Greek letters $\chi$ and $\xi$ to denote computations, $\chi[i]$ stands for the prefix of $\chi$ of length $i$ and $\chi(i)$, for the $i$th computation step.

We denote $L_\omega(\mathcal{A})$ the $\omega$-language accepted by $\mathcal{A}$. It $\mathcal{C}$ is a class of automata we denote respectively by $L(\mathcal{C})$ and $L_\omega(\mathcal{C})$ the classes of languages and $\omega$-languages accepted by a $\mathcal{C}$-automaton. In the same fashion we denote $comp(\mathcal{A})$ and $comp_n(\mathcal{A})$, $comp(\mathcal{C})$ and $comp_\omega(\mathcal{C})$, respectively, the sets of finite and infinite accepting $\mathcal{A}$-runs and classes of sets of finite and infinite accepting runs of $\mathcal{C}$-automata. To formally establish connection between $L(\mathcal{C}_n)$ and $L_\omega(\mathcal{C}_n)$ on the one hand and $comp(\mathcal{C}_n)$ and $comp_\omega(\mathcal{C}_n)$ on the other, we will need to define some predicates and, as auxiliary tools, some recursive numberings. One to code finite runs and another one to code infinite runs.

Let us first consider, for short and w.l.o.g., any state space as a set of integers and the alphabet as the set of integers and the $\omega$ or as an initial segment of $\omega$. Then, for some recursive coding of the plane associating the integer $\langle x, y \rangle$ to each pair $(x, y) \in \omega^2$, let us define the coding:

$$ (x_1, x_2, \ldots, x_n) = \begin{cases} x_n & \text{if } n = 1, \\ (x_1, (x_2, \ldots, x_n)) & \text{otherwise} \end{cases} $$

of $\omega^n$, in a standard way. To access the $i$th component, we use the recursive decoding projection $\pi^n_i$. For short, we will denote $\pi^2_i$ by $\pi_i$. We bear in mind that this gives rise to a recursive coding of $\omega^* = \bigcup_{n \in \omega} \omega^n$ defined as $*\text{-code } (\emptyset) = 0$ and $*\text{-code}$
The decoding projection to access the $i$th letter is thus defined as
\[ u(i) = \pi_1^{[u]}(\pi_2(u - 1)) \]
for $1 \leq i \leq |u|$ and $|u| = \pi_1(u - 1) + 1$.

We code the finite run
\[ q_0 \xrightarrow{x_0} q_1 \xrightarrow{x_1} \cdots \xrightarrow{x_{n-1}} q_n \]
on
\[ *\text{-code}(\langle q_0, x_0 \rangle, \langle q_1, x_1 \rangle, \ldots, \langle q_{n-1}, x_{n-1} \rangle, \langle q_n \rangle) \]
and, finally,
\[ q_0 \xrightarrow{x_0} q_1 \xrightarrow{x_1} \cdots \xrightarrow{x_{n-1}} q_i \xrightarrow{x_i} \cdots \]
on the function $f : \omega \rightarrow \omega$ defined by $f(i) = \langle q_i, x_i \rangle$.
Also, we will make use of a function $[i] : \omega^* \cup \omega^0 \rightarrow \omega$ coding truncated objects at depth $k$ according to the definitions
\[ f[0] = 0 \text{ for } f \in \omega^0, \]
\[ f[i] = *\text{-code}(f(0), f(1), \ldots, f(i - 1)) \text{ for } f \in \omega^0 \text{ and } i \geq 1, \]
\[ u[0] = 0 \text{ and } 0[i] = 0 \text{ for } u \in \omega^* \text{ and } i \in \omega, \]
\[ u[i] = *\text{-code}(u_1, \ldots, u_{\min(i,j)}) \text{ for } u = *\text{-code}(u_1, \ldots, u_j). \]

We note that object $O$ is encoded on code $c$ by $O(c)$. Note here that $\chi(f)[i] = \chi(f[i])$ and $\chi(f)[i] = \chi(f[i])$ and this motivated the use of our sometimes overloaded notation. Using that we are ready to define the above sketched computations and traces, in an effective way. Given:

- Predicate $\text{trans}(f, i, A)$ standing for "$\chi(f)[i]$ is an $A$-transition" may be rewritten as
  \[ (\pi_1(f(i)), \pi_2(f(i)), \pi_1(f(i + 1))) \in \rightarrow_A. \]  \hspace{1cm} (6)

- Predicate $\text{run}(f, i, A)$ standing for "$\chi(f)[i]$ is an $A$-run prefix" may be rewritten as
  \[ \forall j < i (\text{trans}(f, j, A) \land \pi_1(f(0)) \in Q_0^A). \]  \hspace{1cm} (7)

- Predicate $\text{ack}(f, i, A)$ standing for "$\chi(f)[i]$ is an accepting $A$-run prefix" may be rewritten as
  \[ \text{run}(f, j, A) \land \pi_1(f(i)) \in Q_F^A. \]  \hspace{1cm} (8)
We immediately obtain \( L_{o_t}(d_T) = \lim_{n \to \infty} \Sigma^*_n \): any set \( U \) satisfying \( U = L(A) \) for an \( A \subset \varnothing^* \) for some \( A \subset \varnothing^* \) can be expressed as

\[
U = \{ u \in X^* : \exists_{e+\text{-code}} \forall_j (i \leq |v| \land \text{ack}(v, i) \land \pi_2(v(j)) = u(j)) \}
\]

which is easily recognized by a simple Tarski–Kuratowski computation, as being a \( \Sigma^*_n \)-set. Result at the limit is a consequence of completeness of \( \varnothing^*-automata. Reverse \)
inclusion is quite straightforward.

5. Testing processes

Let us first sketch a rudimentary and general interaction product \( A_1 \parallel A_2 \) of two \( \varnothing^*-automata \( A_1 \) and \( A_2 \) by synchronizing positions on a common action. This complete over \( X \cup \{ \tau \} \) (with \( \tau \in X \) denoting an “internal” move) corresponds to the so-called experimental system in [4]. From now on, and unless otherwise stated, by traces, we will mean visible traces defined as \( \eta(L(A)) \) and \( \eta(L_{o_t}(A)) \) where \( \eta(x) = x \) for \( x \in X \) and \( \eta(\tau) = \varepsilon \).

Given \( A_1 = (Q_1, Q_{i0}, Q_{i1}, \rightarrow_1) \) and \( A_2 = (Q_2, Q_{20}, Q_{21}, \rightarrow_2) \), then

\[
A_1 \parallel A_2 = (Q, Q_0, Q_f, \rightarrow)
\]

with

\[
Q_0 = \prod_{i=1,2} (((Q_{i0} \cap Q_{i1}) \times \{ 1 \}) \cup ((Q_{i0} \cap Q_{i1}) \times \{ 0 \})),
\]

\[
Q = \prod_{i=1,2} (Q_i \times \{ 0, 1 \}),
\]

\[
Q_f = \prod_{i=1,2} (Q_i \times \{ 1 \}),
\]

while relation \( \rightarrow \) is satisfying

\[
(q_1, s_1, q_2, s_2) \rightarrow (q'_1, s'_1, q'_2, s'_2) \text{ iff } \begin{cases} (q_1 \xrightarrow{x} q'_1) \land q_2 = q'_2 \text{ or } \\ (q_2 \xrightarrow{y} q'_2) \land q_1 = q'_1 \text{ or } \\ \exists y \in X (q_1 \xrightarrow{y} q'_1) \land (q_2 \xrightarrow{y} q'_2) \land x = \tau \end{cases}
\]

with

\[
s'_i = \begin{cases} 0 & \text{if } s_i = 0 \text{ and } q'_i \in Q_{i1} \backslash Q_{i0}, \\ 1 & \text{otherwise} \end{cases}
\]

for \( i \in \{ 1, 2 \} \).

The definition is motivated in the following way. An interaction product is intended to provide a mechanism to test behaviours of mechanical devices using a device from the same class. It therefore has to reflect the intuition that a run will be an accepting run if and only if each component run is an accepting run of the corresponding component automation. To achieve it properly, it is, obviously not sufficient to use \( Q_{i1} \times Q_{i1} \) of guaranteed states because, remembering \( \Phi_i \) as the guarantee constraint of \( A_i \) \( (i = 1, 2) \), \( \exists_i \Phi_1 \land \exists_i \Phi_2 \) may not be collapsed into \( \exists_i(\Phi_1 \land \Phi_2) \). A classical way
to achieve it is to lift up a flag when the current run goes through a final state and to keep track of the flag position in the current state in a similar fashion as is usually done for finite automata over finitary languages. Unfortunately, when one encodes guarantee constraints into synchronization relation, in such a way the initial state set as well as the transition relation inherit the complexity of $Q_{1f}$ and $Q_{2f}$, causing the class $\mathcal{C}_n$ (over $X \cup \{\tau\}$) not to be closed under interaction product since they would have to be kept in $\Sigma_n^*$, by definition. However, and it is sufficient to serve our purpose, the following predicates

- **trans** $(f, i, \mathcal{P}, \mathcal{S})$ standing for "$\chi_f(i)$ is a $(\mathcal{P}||\mathcal{S})$-transition",
- **run** $(f, i, \mathcal{P}, \mathcal{S})$ standing for "$\chi_f[i]$ is a $(\mathcal{P}||\mathcal{S})$-run prefix",
- **ack** $(f, i, \mathcal{P}, \mathcal{S})$ standing for "$\chi_f[i]$ is an accepting $(\mathcal{P}||\mathcal{S})$-run prefix",

and defined in an obvious but a little bit nasty way, may easily be transformed into $\Delta_n^*$-predicates when $\mathcal{P}$ and $\mathcal{S}$ are $\mathcal{C}_n$-automata, turning the set of visible traces of the interaction product of any two $\mathcal{C}_n$-automata into a $\lim_{\tau} \Sigma_n^*$-set. One may then see the interaction product as a kind of jump operator [8].

In Hennessy’s terminology [4], an experiment or test on the process $\mathcal{P}$ by the tester $\mathcal{S}$, specified here by two $\mathcal{C}_n$-automata, is a sequence of possible interactions between the experimenter and the process. The test criterion is here the $\Sigma_n^*$-predicate specifying the final states of $\mathcal{S}$, and of course the guarantee constraint on the process is the $\Sigma_n^*$-predicate specifying those of $\mathcal{P}$.

To properly define our notion of test, let us introduce some auxiliary definitions. A closed run is any accepting run of $\mathcal{P}||\mathcal{S}$. A closed run is maximal if does not exist any way of strictly extending its maximal $\tau$-prefix toward a closed run. An experiment or test on $\mathcal{P}$ by $\mathcal{S}$ is precisely a maximal closed run. A test is successful if it has an accepting $\tau$-prefix and unsuccessful otherwise.

Let us express all this in a more useful way:

- **Predicate** **closed** $(f, \mathcal{P}, \mathcal{S})$ defined as
  \[
  \exists_i \text{ack}(f, i, \mathcal{P}, \mathcal{S}).
  \]  
  (9)

- **Predicate** **ext** $(f, g, k, \mathcal{P}, \mathcal{S})$ defined as
  \[
  \text{closed}(g, \mathcal{P}, \mathcal{S}) \land f[k] = g[k] \land \forall_{i < k} \pi_2[(f[k](i)) = \tau \land \pi_2(f(k)) \neq \tau].
  \]  
  (10)

- **Predicate** **test** $(f, k, \mathcal{P}, \mathcal{S})$ defined as
  \[
  \text{closed}(f, \mathcal{P}, \mathcal{S}) \land \forall_{g} ((\text{ext}(f, g, k, \mathcal{P}, \mathcal{S})) \Rightarrow \pi_2(g(k)) \neq \tau).
  \]  
  (11)

- **Predicate** **success** $(f, k, \mathcal{P}, \mathcal{S})$ standing for "$\chi_f$ is a successful test whose maximal $\tau$-prefix has length $k$" may be rewritten as
  \[
  \text{test}(f, k, \mathcal{P}, \mathcal{S}) \land \text{ack}(f, k, \mathcal{P}, \mathcal{S}).
  \]  
  (12)

- **Predicate** **failure** $(f, k, \mathcal{P}, \mathcal{S})$ standing for "$\chi_f$ is an unsuccessful test whose maximal $\tau$-prefix has length $k$" may be rewritten as
  \[
  \text{test}(f, k, \mathcal{P}, \mathcal{S}) \land \neg\text{ack}(f, k, \mathcal{P}, \mathcal{S}).
  \]  
  (13)
Now, we tabulate the possible test results on \( P \) by \( \mathcal{E} \) using the symbols \( \top \) to denote a successful computation and \( \bot \) to denote an unsuccessful one. So let Result \((P, \mathcal{E})\) to be defined by

\[
\top \in \text{Result}(P, \mathcal{E}) \text{ iff } \exists f, \exists m \text{success}(f, m, P, \mathcal{E}),
\]

\[
\bot \in \text{Result}(P, \mathcal{E}) \text{ iff } \exists f, \exists m \text{failure}(f, m, P, \mathcal{E}).
\]

A natural way to define equivalence between processes is of course the following one: given \( P, \mathcal{E} \in \mathcal{O}_n \), \( P \) and \( \mathcal{E} \) are equivalent by test (noted \( P \simeq \mathcal{E} \)) if for every \( \mathcal{O}_n \)-process \( \mathcal{E} \), Result \((P, \mathcal{E}) = \text{Result}(\mathcal{E}, \mathcal{E})\).

Clearly, a test is always inferred from a pair of accepting runs of \( P \) and \( \mathcal{E} \) sharing a common trace prefix of visible actions. So, we finally need predicates to define access to the longest common visible trace of test components:

- **twintrace** \((f, w, k, P, \mathcal{E})\) stands for
  
  "\( w \) is the longest common visible trace prefix of the components of the test \( \chi(f) \)
  and \( k \) is the length of the maximal \( \tau \)-prefix of \( \chi(f) \)")

  may be rewritten into

  \[
  \exists f_1, f_2, i, j, k \text{twin}(f, f_1, f_2, i, j, k, P, \mathcal{E}) \land \eta(\pi^2_i(f_1[i])) = w)
  \]

  with \( \eta(\pi^2_i(f_1[i])) = \eta(\pi^2_i(f_1(0))) \cdots \eta(\pi^2_i(f_1(i-1))) \).

- **twin** \((f, f_1, f_2, i, j, k, P, \mathcal{E})\) stands for

  \[
  \text{ack}(f_1, k, P) \land \text{ack}(f_2, \mathcal{E}) \land \text{twincomp}(f, f_1, f_2, i, j, k, P, \mathcal{E}).
  \]

- **twincomp** \((f, f_1, f_2, i, j, k, P, \mathcal{E})\) stands for

  "\( \chi(f[k]) = \phi(f_1[i]) \parallel \psi(f_2[j]) \)"

It could be nasty to go further in that way, but the informal description of this last predicate provided below will be sufficient to evaluate its logical complexity: given three strings, one has to verify at the first stage that one can recover \( \chi(f)(0) \) from \( \phi(f_1)(0) \) and \( \psi(f_2)(0) \) by means of interaction rules. One may suppose now that at the beginning of a stage in the process, one verified the recovering of \( \chi(f)[k'] \) from \( \phi(f_1)[i'] \) and \( \psi(f_2)[j'] \), then, in the current stage, one checks the recovering \( \chi(f)(k') \) from \( \phi(f_1)(i') \) and \( \psi(f_2)(j') \) by means of interaction rules. The whole process needs \( k \) stages. So, **twincomp** is a \( \Delta^\ast_{n+4} \)-form. We will need this characterization to obtain the next result which appears to be the cornerstone to prove that \( \mathcal{O}_n \) has not the separation property.

**Lemma 5.1.** For \( n \geq 1 \), given an \( \mathcal{O}_n \)-process \( P \) and an \( \mathcal{O}_n \)-tester \( \mathcal{E} \), the set \( S_{\omega_0} \) and \( F_{\omega_0} \) of \( P \)-traces resulting in a successful and unsuccessful test respectively at \( \lim_{\omega} \Sigma^\ast_{n+2} \)-sets.

**Proof.** We first prove that set \( S \) of finite succeeding traces, defined as

\[
\{w \in X^* : \exists f \perp_k (\text{success}(f, k, P, \mathcal{E}) \land \text{twintrace}(f, w, k, P, \mathcal{E}))\},
\]

is a \( \Sigma^\ast_{n+2} \)-set.
Eq. (18) involves a quantifier ranging over an infinite object. Here we can simplify because this quantifier is easily eliminated [8, Theorem 16-VII]: the second-order predicate in Eq. (18) can be transformed into

$$\exists u \forall k (k \leq |u| \land \text{success}(u, k, \mathcal{P}, \mathcal{E}) \land \text{twintrace}(u, w, k, \mathcal{P}, \mathcal{E}))$$

(19)

where $u$ is an $*$-code and it easily follows from the Tarski–Kuratowski algorithm that Eq. (19) is a $\Sigma^*_n$ predicate:

- **closed**$(u, \mathcal{P}, \mathcal{E}) \in \Delta^*_n$:

$$\exists i \leq |u| \land \text{ack}(u, i, \mathcal{P}, \mathcal{E})$$

(20)

- **ext**$(u, v, k, \mathcal{P}, \mathcal{E}) \in \Delta^*_n$:

$$\text{closed}(v, \mathcal{P}, \mathcal{E}) \land u[k] = v[k] \land \forall i < k \pi^*_i(u[i]) = \tau \land \pi^*_i(u(k)) \neq \tau$$

(21)

- **test**$(u, k, \mathcal{P}, \mathcal{E}) \in \Pi^*_n$:

$$\text{closed}(u, \mathcal{P}, \mathcal{E}) \land \forall t \left((k \leq |t| \land \text{ext}(u, t, k, \mathcal{P}, \mathcal{E})) \Rightarrow \pi^*_t(t(k)) \neq \tau\right)$$

(22)

- **success**$(u, k, \mathcal{P}, \mathcal{E}) \in \Pi^*_n$:

$$k \leq |u| \land \text{test}(u, k, \mathcal{P}, \mathcal{E}) \land \text{ack}(u, k, \mathcal{P}, \mathcal{E})$$

(23)

- **twin**$(u, u_1, u_2, i, j, k, \mathcal{P}, \mathcal{E}) \in \Delta^*_n$:

$$\text{ack}(u_1, k, \mathcal{P}) \land \text{ack}(u_2, k, \mathcal{E}) \land \text{twincomp}(u, u_1, u_2, i, j, k, \mathcal{P}, \mathcal{E})$$

(24)

- **twintrace**$(u, w, k, \mathcal{P}, \mathcal{E}) \in \Sigma^*_n$:

$$\exists u_1, u_2, i, j \left(\text{twin}(u, u_1, u_2, i, j, k, \mathcal{P}, \mathcal{E}) \land \eta(\pi^*_i(u_1[i])) = w\right)$$

(25)
By substitution, we then find that Eq. (19) is a $\exists(\Pi_{n+1}^* \land \Sigma_{n+1}^*)$-form, that is to say a $\Sigma_{n+2}^*$-form. The fact that set $S_\omega$ of succeeding traces is a $\lim_{\Sigma^*_n}$ set trivially follows from completeness of $\mathcal{P}$.

In a similar fashion for the failure part: we have to estimate the logical complexity of set $F$ of finite failing traces, defined as

$\{w \in X^*: \exists f \exists k (\text{failure}(f, k, \mathcal{P}, \mathcal{E}) \land \text{twintrace}(f, w, k, \mathcal{P}, \mathcal{E}))\}$. (26)

The quantifier on infinite objects appearing in Eq. (26) may easily be eliminated as above and turned into

$\{w \in X^*: \exists u \exists k \leq |u| (\text{failure}(u, k, \mathcal{P}, \mathcal{E}) \land \text{twintrace}(u, w, k, \mathcal{P}, \mathcal{E}))\}$ (27)

where $u$ is a *-code. By substitution firstly of Tarski–Kuratowski results computed in the success part and secondly of the following,

• failure$(u, k, \mathcal{P}, \mathcal{E}) \in \Pi_{n+1}^*$:

$$\Delta_T^{\Pi_{n+1}^*} \land \Delta_{\Pi_{n+1}^*}^{\Pi_{n+1}^*} \land \Delta_{\Pi_{n+1}^*}^{\Pi_{n+1}^*} \land \Delta_{\Pi_{n+1}^*}^{\Pi_{n+1}^*}$$ (28)

we find that Eq. (27) is also of $\exists(\Pi_{n+1}^* \land \Sigma_{n+1}^*)$-form and the rest of the proof remains the same as the success part. ☐

6. The separability result for processes

We will now state the exact notion of separability for processes:

Definition 6.1. A class $\mathcal{C}$ of processes will be said to be separable by test if for any pair $(\mathcal{P}, \mathcal{E})$ in $\mathcal{G}$ there exists a process $\mathcal{B} \in \mathcal{C}$ such that

$$\text{Result}(\mathcal{P}, \mathcal{E}) \neq \text{Result}(\mathcal{B}, \mathcal{E}).$$

To answer the question of separability of $\mathcal{C}_n$, we shall express $\simeq$ in terms of the more primitive relations may and must [4]:

$\mathcal{P}$ may $\mathcal{E} \iff \top \in \text{Result}(\mathcal{P}, \mathcal{E}),$

$\mathcal{P}$ must $\mathcal{E} \iff \bot \notin \text{Result}(\mathcal{P}, \mathcal{E}).$

This enables us to define preorder relations

$\mathcal{P} \subseteq_{\text{MAY}} \mathcal{B} \iff \forall \mathcal{E} \in \mathcal{C}_n (\mathcal{P} \text{ may } \mathcal{E} \Rightarrow \mathcal{B} \text{ may } \mathcal{E}),$

$\mathcal{P} \subseteq_{\text{MUST}} \mathcal{B} \iff \forall \mathcal{E} \in \mathcal{C}_n (\mathcal{P} \text{ must } \mathcal{E} \Rightarrow \mathcal{B} \text{ must } \mathcal{E}),$

inducing equivalence relations $\simeq_{\text{MAY}}, \simeq_{\text{MUST}}$ in the usual way.
We immediately have [4]

\[ \mathcal{P} \simeq \mathcal{Q} \text{ iff } \mathcal{P} \simeq_{\text{MAY}} \mathcal{Q} \text{ and } \mathcal{P} \simeq_{\text{MUST}} \mathcal{Q}. \]

We will now apply the previous results in order to establish the some \( \mathcal{O}_n \)-processes discriminated throughout their set of traces (i.e. throughout some finite or infinite sequences of visible behaviours) are nevertheless not distinguishable by test.

**Theorem 6.2.** For \( n \geq 1 \), \( \mathcal{O}_n \) is not separable by test.

**Proof.** By Theorem 3.2., we know that, for each \( n \), \( U \) and \( V \) are not separable in \( \lim \Sigma_i^* \) for \( U = X^\omega \), and \( V = U \setminus \{ \beta \} \). Given deterministic \( \mathcal{P} \), \( \mathcal{Q} \in \mathcal{O}_n \) such that \( L_\omega(\mathcal{P}) = U \), \( L_\omega(\mathcal{Q}) = V \).

Since \( \text{Comp}_\omega(\mathcal{Q}) \subseteq \text{Comp}_\omega(\mathcal{P}) \), we immediately have

\[ \mathcal{Q} \subseteq_{\text{MAY}} \mathcal{P} \text{ and } \mathcal{P} \subseteq_{\text{MUST}} \mathcal{Q}. \]

One may now suppose, toward a contradiction, that

\[ \mathcal{P} \not\subseteq_{\text{MAY}} \mathcal{Q} \text{ or } \mathcal{Q} \not\subseteq_{\text{MUST}} \mathcal{P}. \tag{29} \]

The left-hand side of Eq. (29) implies that there exists an \( \mathcal{O}_n \)-tester whose only \( \mathcal{P} \)-computation resulting in a successful test is the run over \( \beta \), contradicting Lemma 5.1 since there is no singleton in \( \lim \Sigma_i^* \).

In a similar way, the right-hand side of Eq. (29) means that for some \( \mathcal{O}_n \)-tester the only \( \mathcal{P} \)-computation resulting in an unsuccessful test is the run over \( \beta \) contradicting Lemma 5.1 again since there is no singleton in \( \lim \Sigma_i^* \Sigma_{n+1}^* \) and this completes the proof. \( \Box \)

We will conclude this section, by highlighting that our \textit{must} definition is clearly less general than Hennessy’s one in the following sense: Hennessy’s \textit{must} gets capabilities to manage with some infinite divergent computations.

**7. Logical complexity results**

We now postpone any further analysis through one among the many recursive process specification formalisms (e.g. see [2] for such an analysis applied to Milner’s CCS) and take rather advantage of our framework to establish some results concerning the logical complexity of various notions of test equivalence relations on \( \mathcal{O}_n \).

Assume a Gödel numbering of all recursively enumerable languages over an alphabet \( X \). This yields recursive numberings for each class \( \Sigma_i^* \) and each class \( \Pi_i^* \) called respectively \( \Sigma_i^* \)-index and \( \Pi_i^* \)-index [8] (Section 14.2) and, of course, recursive numberings for each class \( \lim \Sigma_i^* \Sigma_{n+1}^* \) and each class \( \lim \Pi_i^* \Pi_n^* \). From that numbering, we could easily imagine recursive numberings for each class of \( \mathcal{O}_n \)-processes as well. We refer the reader to [6] where such indices are built up for more general classes of
recursive automata. Let us call such an index an \textit{n-index}. Writing \(\delta_{(i)}\) to mean that \(i\) is the \(n\)-index of \(\mathcal{D}\) (for \(n\) fixed), the above equivalences are encoded into the following predicates:

\[
\mathcal{P} \equiv_{\text{MAY}} \mathcal{D} \iff \forall i (\mathcal{P} \text{ may } \delta_{(i)} \iff \mathcal{D} \text{ may } \delta_{(i)}), \tag{30}
\]

\[
\mathcal{P} \equiv_{\text{MUST}} \mathcal{D} \iff \forall i (\mathcal{P} \text{ must } \delta_{(i)} \iff \mathcal{D} \text{ must } \delta_{(i)}). \tag{31}
\]

Using \(n\)-indices, the testing problem \(\text{TEST}_n\) can be formulated now as

\[
\text{TEST}_n = \{(i, j) : \mathcal{P}_{(i)} \equiv_{\text{TEST}} \mathcal{D}_{(j)} \text{ and } n\text{-codes } i, j\}.
\]

Completing Tarski–Kuratowski computations of Lemma 5.1, we obtain:

\textbf{Theorem 7.1.} For \(n \geq 1\), \(\text{TEST}_n\) is in \(\Pi^{n+3}_{n+3}\).

\textbf{Proof.} We have to show that given \(\mathcal{P}, \mathcal{D} \in \mathcal{C}_n\), then \(\mathcal{P} \equiv_{\text{TEST}} \mathcal{D}\) is a \(\Pi^{n+3}_{n+3}\)-predicate for \(\text{TEST} \in \{\text{MAY, MUST}\}\).

We already know, from Lemma 5.1, that \(\mathcal{P} \text{ may } \delta\) is given by a \(\Sigma^{n+2}_{n+2}\)-form. So, \(\mathcal{P} \equiv_{\text{MAY}} \mathcal{D}\) is a \(\forall (\Sigma^{n+2}_{n+2} \Rightarrow \Sigma^{n+2}_{n+2})\)-form that is a \(\Pi^{n+3}_{n+3}\)-form and \(\mathcal{P} \equiv_{\text{MAY}} \mathcal{D}\) is therefore a \((\Pi^{n+3}_{n+3} \wedge \Pi^{n+3}_{n+3})\)-form which is still a \(\Pi^{n+3}_{n+3}\)-form.

For the \textit{MUST} part, using again Lemma 5.1, we compute \(\mathcal{P} \text{ must } \delta\) as a \((-\Sigma^{n+2}_{n+2})\)-form which is a \(\Pi^{n+2}_{n+2}\)-form. So, \(\mathcal{P} \equiv_{\text{MUST}} \mathcal{D}\) is a \(\forall \Pi^{n+2}_{n+2} \Rightarrow \Pi^{n+2}_{n+2}\)-form that is a \(\Pi^{n+3}_{n+3}\)-form and the result for \(\mathcal{P} \equiv_{\text{MUST}} \mathcal{D}\) follows in the same way as the \textit{MAY} part. \(\square\)

\section{8. Conclusion and future works}

In this work, precise connections have been made between a hierarchy of “effectively generated” open \(\omega\)-languages, not bounded by any arithmetical class of \(\omega\)-languages, and an hierarchy of communicating recursive automata. Starting from a universal model, the jump from one degree to the next one arises when automata communicate together.

It has been established that no class of the \(\omega\)-language hierarchy has the separation property and, derived from this result, that no class of the automation hierarchy is separable by test. The testing problem for degree \(n\) of the automation hierarchy has been clearly formulated and proved to be in \(\Pi^{n+3}_{n+3}\).

To conclude the paper, we point out some open problems. The first one is related to the construction of concurrency models with the ability to discriminate at all time between pairs of processes that differ in their respective sets of infinite visible behaviours but at a reasonable complexity cost: \(\Delta^1_1\), a class having the separation property since it is closed under complementation, has not yet an operational definition while CCS may be regarded as such a definition for \(\Sigma^1_1\).
The second problem is related to the separability problem. We bear in mind that, as pointed out in Section 3, any \(\omega\)-language class obtained from \(\omega\)-Kleene closure of a language class containing context-free languages and closed under any usual operation (finite union, intersection with regular language, concatenation product, Kleene closure, alphabetic morphisms and their inverses) is not separable. We could obtain an extension to this result in proving that the \(\omega\)-Kleene closure of deterministic context-free languages, a language class having all the required properties but not containing context-free languages, does not possess the separation property. There is work in progress to solve a Darondeau conjecture proving this result.

Finally, concerning complexity issues, note that even if we know that \(\{\text{TEST}_n\}\) forms a hierarchy and that \(\text{TEST}_n \subseteq \Pi^*_n\), it is not sufficient to prove that \(\{\text{TEST}_n\}\) forms a hierarchy as well. However we conjecture that \(\text{TEST}_n\) is \(\Pi^*_n\)-complete.

Acknowledgements

I thank Andrè Arnold and Philippe Darondeau for carefully reading previous versions of this text and for their numerous comments, remarks and suggestions.

References