# Steps or stages for incremental learning

Dominique Luzeaux
Eric Martin
ETCA/CREA/SP; 16 bis, av. Prieur de la Côte d'Or; 94 114 Arcueil Cedex; France
E-mail: luzeaux@etca.fr, martin@etca.fr

## Introduction

Formal work on learning consists in an ever growing class of paradigms, which may be strongly related or may only share very few points. Our research is concerned with the following issue: how to learn to control a physical system. This is a major field of interest for AI, and many proposals have emerged to solve this problem. But the most part of it is mainly empirical work and is not concerned with the theoretical work we mentioned above. In this paper we would like to describe a few methodological principles that are valuable with respect to two issues: on the one hand we can find techniques that implement them and we can eventually write programs that learn to control a system; on the other hand we can study paradigms that take those principles as axioms. Such paradigms are very different from the usual ones and give a new insight on incremental learning.

In the first section we will present in a rather informal way the methodological principles we had to introduce in order to tackle our learning problem. We will show how they *differ* from usual principles and why their introduction was *necessary*. A close investigation of the relationship between the flow of data and the hypotheses proposed by the learner will lead to the notion of *learning stages rather than learning steps*. In the same way as we had to adapt learning methodologies to deal with a control problem, we had to find within control theory a framework within which learning was possible. This will be discussed in the second section. In the third section, we will give one possible formalization of the methodological principles of the first section, in order to formulate a result which shows precisely why the concept of stage is an important one.

## 1 How to deal with real environments?

### 1.1 Learning paradigms

As there is no general definition of what learning is, it is hopeless to propose a unified framework for all the paradigms of learning. Nevertheless, we will first describe a few characters that are common to almost all existing paradigms. These preliminary remarks will be very useful to point out the specificity of control and of our methodology in the formal work on learning.

A *learner* is confronted to a *learning problem*. We conceive the learner to be a function; we do not need to define right now what a learning problem is. The learner is given an

infinite sequence of data; we call this sequence an *environment*. He associates a value with finite initial segments of the environment (not necessarily with all initial segments); these values are called *hypotheses* [OSW86]. An environment depends on the learning problem and on the learner; there are many possible environments. A criterion of success is a property of the environment and of the sequence of associated hypotheses. We say that *the learner learns the learning problem* if the criterion of success is true for every environment and sequence of associated hypotheses.

More formally, an environment function (respect. a learning function) is a function $E$ (respect. $H$) such that[1]:

i) $E : \bigcup_{n\in\mathbf{N}}(\mathbf{N}^n \times (\mathbf{N}\cup\{\uparrow\})^n) \rightarrow \mathbf{N}$

ii) $H : \bigcup_{n\in\mathbf{N}}\mathbf{N}^n \rightarrow \mathbf{N}\cup\{\uparrow\}$

Let $(\alpha_n)_{n\in\mathbf{N}}$ be a sequence, $n$ an integer; we denote the finite sequence $(\alpha_0,\ldots,\alpha_n)$ by $\overline{\alpha_n}$. We define the following sequences:

*i)* $e_n = E(\overline{e_{n-1}}, \overline{h_{n-1}})$

*ii)* $h_n = H(\overline{e_n})$

A paradigm $\mathcal{P}$ defines, for each learning problem $LP$, two predicates $P(E,H)$ and $R(E,H)$, where $E$ is an environment function and $H$ a learning function. $P(E,H)$ means that the paradigm intends considering learners who propose hypothesis $h_n$ when they are given the sequence of data $\overline{e_n}$, for any $n \in \mathbf{N}$, and environments where $e_n$ is the $n^{th}$ datum that is given to the learner, for any $n \in \mathbf{N}$. $R(E,H)$ means that the paradigm intends considering successful a learning process where the learning function is $H$ and the environment function is $E$. The learning problem $LP$ is *learnable in the paradigm* if:

$$(\exists H)(\forall E)(P(E,H) \rightarrow R(E,H)).$$

Let us make the following remark. The datum $e_n$ depends partially on $\overline{h_{n-1}}$, which means that the hypothesis the learner proposes can somehow have an effect on the data that follow; this phenomenon is called "*reactivity of the environment*". Formal work on learning often makes the assumption that there is no reactivity of the environment, that is to say $e_n$ can be written as $E'(\overline{e_{n-1}})$. Various types of so-called reactivity have already been considered in some other formal work, but they do not conform to the framework we have given: more precisely, the environment is generally assumed to be either an oracle or an adversary. For instance, situations are considered where the learner does not only propose hypotheses, but may also propose a value, say $v$, and the environment gives a response to $v$. The kind of reactivity that our formulation intends to capture is quite different: we do not want to assume that the learner has the ability to "ask" the environment, or to "reply" to it. But we are dealing with situations where the state of the learner (i.e. the latest hypothesis he has proposed) necessarily "constrains" the kind of data he can get while he is in this particular state.

Let us remark too that we assumed the data and the hypotheses could be coded by integers.

[1] Let $\varphi$ be a partial function; when $\varphi(x)$ is undefined, we write $\varphi(x) = \uparrow$.

## 1.2 Control and reactivity

Now, we will characterize more precisely the concepts we have just introduced, by analyzing the problem of learning to control a system. Actually our problem is to find a controller (as will be defined formally in the next section). We will assume a controller can be described in a given language. For instance, a controller may be a finite set of rules, and a rule could be: "if this condition is true, then do this action". We will make this more precise later on. Here, suffice it to say that the set of controllers is denumerable, and that we can effectively list all the controllers (thanks to their description, by lexicographical order for instance). Let us consider a system to be controlled. We assume for now that a controller is either satisfactory or unsatisfactory, and that there exist satisfactory controllers. The aim of the learning procedure is to find eventually a satisfactory controller.

We are dealing with discrete control, which means that measures of the parameters — inputs (commands) and outputs (observables) — of the system to control are made at fixed intervals of time. A datum is an evolution, that is to say an initial state, and the sequence of the measures of the parameters on some while. This concludes the definition of hypotheses and data. Now we have a methodological principle: *data (evolutions) are generated by the latest hypothesis (controller) the learner has proposed.* This is of course a remarkable kind of reactivity of the environment, in which the environment can be seen as an oracle of a very particular kind: if the length of an evolution is $n$, the learner asks the environment what the values of the parameters are, then he acts according to the controller he has chosen to use, then he asks the same question, then he acts...; a datum is the result of $n$ questions to the environment, and of $n$ answers. Nevertheless, the questions the learner can ask cannot be independent from the hypotheses he emits, as is assumed classically: they are *always determined* by these hypotheses!

Why can a given controller determine many evolutions? First, the length of an evolution may vary; second, many rules from the same controller can be fired in a given situation (one may be selected at random); third, initial evolutions may vary too. So we see that for each controller $C_i$, there is a set of evolutions $E_i$ that can be generated by this controller. The learner is not supposed to fix initial situations, length of evolutions or choice between admissible rules, so reactivity of the environment just means that data have to belong to $E_i$ when controller $C_i$ is used. Let us note too that the learner is not supposed to decide how many data (evolutions) he will collect with the latest controller he has proposed.

This points out that the relationship between the learner and the environment is more complex than it is in other works. Usually, the predicate $P(E, H)$ would have just been written $P'(E)$: there is a set of potential environments, and the learner has to adapt to all of them. But because of the kind of reactivity we have described, we must write $P(E, H)$, which means that the learner has to adapt to the environment *and* that the environment has to be adapted to the manner the learner is learning! This has very important consequences on the whole formalization of learning, and it is required by reactivity.

## 1.3 Control ability and system identification

We have to suggest now what the predicate $P(E, H)$ might be. Let us consider an integer $N$ such that $h_N \neq \uparrow$. At the $N^{th}$ *step*, the learner emits hypothesis $h_N$, which means that he will use controller $C_{h_N}$, and evolutions generated at steps $p > N$ will belong to $E_{h_N}$ as

long as $h_p = \uparrow$. At some time $N' > N$, the learner will propose a new hypothesis $h_{N'}$. The interval of time $[N, N']$ constitutes a *stage*; it is characterized by the fact that $h_N$ and $h_{N'}$ are different from $\uparrow$, while the $h_p$, $N < p < N'$, are equal to $\uparrow$.

With our methodology, in order to find a controller, we first try to find a model of the system to control (this corresponds to an *identification* of the system). Then we use this model to propose a controller. To complete the identification, the learner analyzes the finite sequence of evolutions collected during a learning stage. This analysis gives him some information on the system (for instance a statistical or qualitative model [LZ90]). We assume the information that can be discovered with a given controller follows a *compactness* principle: for each integer $i$, there is a finite sequence $\sigma$ of elements of $E_i$ such that for any finite sequence $\sigma'$ of elements of $E_i$ containing $\sigma$, the information yielded by $\sigma$ is equal to the information yielded by $\sigma'$. In other words, $\sigma$ contains *all* the information that can be yielded by $C_i$. More formally, if the learner has really proposed an hypothesis at step $N$, we can write:

$$(\exists e_{N+1}, \ldots, e_{N+p})(\forall e'_{N+1}, \ldots, e'_{N+p'})(\{e_{N+1}, \ldots, e_{N+p}\} \subseteq \{e'_{N+1}, \ldots, e'_{N+p'}\} \subseteq E_{H(\overline{e_N})})$$
$$\rightarrow (H(\overline{e_N}, e_{N+1}, \ldots, e_{N+p}) = H(\overline{e_N}, e'_{N+1}, \ldots, e'_{N+p}))$$

We see that the concept of stages, not the concept of steps, is the right one in this case, because we expect a stage to bring the learner the whole information that is "contained" in the associated controller. Hence the learner must be provided with a good set of data at each stage if he has to learn as we expect him to learn. This is better said than done, and it is undoubtly a challenge for incremental learning. We shall soon see why.

Let $E$ and $H$ be such that the predicate $P(E, H)$ is defined by the above equation; we can then define the function $\overline{H}$ in the following way:

$$\overline{H}(0) = H(\epsilon) \text{ ($\epsilon$ is the empty sequence)}$$

$$\overline{H}(n) = H(\overline{e_{\alpha(n)}})$$
where $\alpha(n)$ is the smallest $i$ such that $|\{j, 0 \leq j \leq i \mid H(\overline{e_j}) \neq \uparrow\}| = n$

so that $\overline{H}(n)$, $n \in \mathbf{N}$, is the hypothesis proposed by the learner at stage $n$. Note that in our general formulation, the function $H$ analyses all the data that have appeared, whether they belong to the same stage or to previous ones. In fact, we expect $H$ not to be so general, for instance ($n \geq 1$):

*i)* $H(\overline{e_{\alpha(n)}}) = G_1(\overline{H}(n-1), e_{\alpha(n-1)+1}, \ldots, e_{\alpha(n)})$

*ii)* $H(\overline{e_{\alpha(n)}}) = F(\overline{H}(n-1), G_2(e_{\alpha(n-1)+1}, \ldots, e_{\alpha(n)}))$

*iii)* $H(\overline{e_{\alpha(n)}}) = G_3(e_{\alpha(n-1)+1}, \ldots, e_{\alpha(n)})$

In all these cases, the learner takes into account the data of the current stage, and possibly (cases *i)* and *ii)*) the hypothesis of the previous stage.

Let us now discuss the predicate $R(E, H)$. In the existing literature things are considered rather differently with respect to this matter. There exist two main approaches: "convergence in the limit" models [Gol67], and "probably approximately correct" models [Val84]. Only the first fits into the framework we propose. A particularly interesting case is provided with a function $H$ which is of the kind *iii)* above, because it can be motivated by a methodological principle: *a learner analyzes the finite set of evolutions that constitutes the data of a given stage in some predefined way and does not take into account*

*the previous data and hypotheses, because we think the more satisfactory a controller, the better the hypothesis (controller) the learner will infer from it.* Learning is expected to conform to a progression principle. This concept will be detailed in the last section. Here, we want just to point out that in case *iii*), learning converges iff the same hypothesis is proposed twice consecutively. Let us assume that the set of hypotheses can be separated into two classes: satisfactory hypotheses and unsatisfactory hypotheses. If the aim of the learner is to find eventually a good hypothesis, learning must stop at some stage, and we know when it stops: when the same hypothesis is proposed twice consecutively.

Under these criteria, it is easy to exhibit a set of learning problems that is learnable, but such that the set of data to be collected during a stage, for all stages and all learning problems of the class, is necessarily not recursive. This raises the problem of constraining the environment to be adapted to the learning program used, whatever this one may be. We underlined that the kind of reactivity of the environment that has to be taken into account, if we want to deal with control problems, puts stages against steps. A step is of length one, a stage is of length $n$. This result shows that $n$ cannot always be calculated in an effective manner. The question "what is a stage?" is related to the question "what is to be expected from the environment with respect to the data it will provide to the learning program?". We must ask this question because we will see in the last section that all the learning programs taken out of a large class may need more than one stage to learn to control systems.

## 2 Incremental learning and incremental control

Control theory deals with systems which can be modeled as a relation defined in $I \times O$ where $I$ is the *input* set and $O$ the *output* set. As we will deal from now on with discrete systems, we introduce the following notations: the input at time $k$ is denoted by $u_k$ and the output at time $k$ by $y_k$. One of the main problems addressed in control theory is to determine the inputs $(u_k)_{k\in\mathbf{N}}$ such that desired outputs $(y_k)_{k\in\mathbf{N}}$ can be obtained.

As we want to learn to control a system, we have to look for a convenient way to express this relation between inputs and outputs.

Quite a few experiments seem to point at the following observation: when faced by an unknown or ill-known system, human experts look often for a rule-based representation of the system like: "if I observe that state, then I act this way". Of course the concept of state at this level is informal. Furthermore the action is often incremental: this input will be increased, that one will be decreased or left untouched.

The preceding discussion is entirely informal and does not pretend to have found the behavior model of any human; it is only based on various experiments (cutting metal plaques with a laser robot [FKB+85], driving a car [Fou90, Luz92a], landing a plane [HLG91],...).

The formal tool resulting from that discussion is *rule-based incremental control* [Luz91]. An incremental control law relates $u_{k+1}$ to $u_k$ by:

$$u_{k+1} = u_k + \epsilon_k \Delta$$

where $\Delta$ is a non null positive real and $\epsilon_k$ is an integer in $\{-m, \cdots, +m\}$ where $m$ depends only on the system to control. A rule-based incremental controller is a finite set of rules that have the following form: `if` *some condition on* $y_k$ `then` *choose* $\epsilon_k$. In other words, we have:

$$\epsilon_k = S(g(y_0, \cdots, y_k, u_0, \cdots, u_k))$$

where $g$ is a computable function and $S$ an integer valued step function.

This definition is very general and usually $g$ depends only on $y_k$ and $u_k$ (for instance for car parking [Luz92b]).

This formulation of rule-based incremental control satisfies the assumptions made in the previous section: every controller is a computable function, all the controllers can be effectively listed, and at each time $k$ an action is chosen among a finite set of possible actions.

Of course the main question is then: is that type of control efficient? The answer is: yes! We will only state the most important results here; proofs and further results can be found in the references:

- any controllable linear system, time-invariant or time-varying, can be stabilized by a rule-based incremental controller, either around an equilibrium or around a reference trajectory [Luz92b]
- non linear systems linear in control (like bilinear systems, mobile robots and other non holonomic systems,...) can be exponentially asymptotically stabilized around a reference trajectory by a rule-based incremental controller under some complete controllability assumptions [Luz92b]
- the range of $\epsilon_k$ depends on the system to control; some necessary inferior bounds are given in [Luz91]
- car-like mobile robots can be completely controlled by rule-based incremental controllers in any connected environment with any distribution of obstacles [Luz92a]

Thus incremental control provides us with an efficient tool within control theory; furthermore its expression is easy enough to comply with learning principles.

# 3 A framework for learning in control

Our aim is to show that in a situation of the kind described before, the concept of stage cannot be eliminated. To state this result precisely, we need to formalize the points we described informally in section 2.

## 3.1 Learning problems and learning functions

An *evolution* is a finite sequence of 2 actions, starting from an initial situation, so it can be represented by a (possibly empty) finite sequence of 1's and 2's. We denote the set of all evolutions by $\mathcal{E}$. We consider a recursive coding of $\mathcal{E}$ upon $\mathbb{N}$, and if $x_i$ (for $1 \leq i \leq n$) is equal to 1 or 2, $< x_1, \ldots, x_n >$ represents the $n$-tuple or its image under this coding, according to the context. We write $x \sqsubseteq y$ if $x$ is an initial segment of $y$, and $x \sqsubset y$ if it is a proper one.

A *set of rules* is a set of evolutions a controller can produce when applied to a particular system, starting from an initial situation. As we have no definition of what a 'system' is, we place only two constraints: first, if a set of rules can produce an evolution, it can also produce all of its sub-evolutions (it has a tree structure); second, it is recursively enumerable (r.e.). More formally:

**Definition 3.1** *A set of rules is a non empty r.e. set $R$ of elements of $\mathcal{E}$ such that:*

$$(\forall x \in R)(\forall y \in \mathcal{E})(y \sqsubseteq x \rightarrow y \in R).$$

A set of rules $R$ is *total* if a rule can always be fired at any point, i.e.: $(\forall x \in R)(\exists y \in R)(x \sqsubset y)$.
For every r.e. set $P$, we write $P^{\diamond}$ for the smallest set of rules containing $P$.
Of course initial situations may vary, but instead of associating one tree of evolutions $T_i$ with each of them, we find it convenient to consider a single tree which 'gathers together' all the $T_i$, and which agrees legitimately with the preceding definition.

The system has viability limits, and it has to be controlled so that it does not reach them. In other words, an evolution may or may not be satisfactory, and a controller tries to rule out those which are not. A *set of constraints* contains the unsatisfactory evolutions. This time, we can place three constraints: first, this set contains all super-evolutions of every evolution it contains; second, if it does not contain a given evolution, it does not contain at least one strict super-evolution of it (we rule out 'catastrophic' evolutions); third, it is r.e. (we can effectively decide whether an evolution reaches a viability limit, and the catastrophic evolutions are recursively generated from them). More formally:

**Definition 3.2** *A set of constraints is an r.e. set $C$ such that:*

*i)* $(\forall x \in C)\ (\forall y \in \mathcal{E})\ (x \sqsubseteq y \rightarrow y \in C)$;

*ii)* $(\forall x \in \mathcal{E})\ [(\forall y \in \mathcal{E})\ (x \sqsubset y \rightarrow y \in C) \rightarrow x \in C]$.

For every r.e. set $P$, we write $P^{\otimes}$ for the smallest set of constraints containing $P$.

Our learning theory will take evolutions as examples; satisfactory evolutions are positive examples, and unsatisfactory ones are negative examples.

The controllers as they have been introduced in the previous section have a definite syntax, and we can effectively enumerate them (for instance, according to the number of rules, and by lexicographical order). Therefore, corresponding to the system to be controlled, there is a recursive enumeration of the sets of rules. We now have enough terminology to define the concept of a *learning problem.* We just need an acceptable indexing of the recursive functions [Rog67]: $(\varphi_i)_{i \in \mathbb{N}}$; we denote the domain of $\varphi_i$ by $W_i$.

**Definition 3.3** *A learning problem is a couple $[c, r]$, where $c$ is an index of a set of constraints, and $r$ is an index of a recursive function which enumerates indexes of sets of rules:*

i) $W_c = W_c^{\otimes}$; ii) $(\forall i \in \mathbb{N})(W_{\varphi_r(i)} = W^{\diamond}_{\varphi_r(i)})$.

The intended interpretation of $W_{\varphi_r(i)}$ is the set of evolutions that the $i^{th}$ controller can produce. The learner is faced with the problem of finding a controller (an $i$) such that $W_{\varphi_r(i)} \cap W_c$ is 'not too big' – if possible empty – and $W_{\varphi_r(i)} \cap \overline{W_c}$ is 'not too small' (we will give precise definitions for these somewhat vague concepts).

A *hypothesis* is an integer: when the learner proposes hypothesis $i$, it means that he intends to use the $i^{th}$ controller. The only examples available to him, under that choice, are those belonging to $W_{\varphi_r(i)}$. After the learner has proposed hypothesis $i$, he observes a number of evolutions from $W_{\varphi_r(i)}$ generated by the $i^{th}$ controller, and we have already emphasized that we do not necessarily assume that this number is fixed.

At each stage, the learner studies the finite set of evolutions he has been given. We stated that for each set of rules $R$, there exists a finite subset $D$ of $R$ such that each subset of $R$ which contains $D$ leads to the same hypothesis as $D$ does. To express this compactness principle, we define what we call a *stabilized function.* It is the formal counterpart of the work the learner does with the evolutions he has obtained during a learning stage. More precisely, it is relative to one given problem, and it is to be seen as a function which converts potential experiences into hypotheses. Let us write $D_x$, where $x$ is any integer, for the set $\{x_1, \ldots, x_n\}$ if $x = 2^{x_1} + \ldots + 2^{x_n}$, and for the empty set if $x = 0$. Let $\mathcal{P}$ be a subset of the class of all subsets of $\mathbb{N}$ (we intend $\mathcal{P}$ to be interpreted as the class of all the sets of rules of a learning problem).

**Definition 3.4** *A recursive function $f$ is* stabilized *(on $\mathcal{P}$) if it satisfies the following condition:*

$$(\forall P \in \mathcal{P})(\exists y \in \mathbb{N})((D_y \subset P) \wedge (\forall y' \in \mathbb{N})(D_y \subset D_{y'} \subset P \rightarrow f(y) = f(y'))).$$

*For all $P$ in $\mathcal{P}$, every $y$ in $\mathbb{N}$ (or every $D_y$) which satisfies the above condition, is called a* stable point *of $f$ on $P$, and $f(y)$ is the* limit *of $f$ on $P$.*

We write: $\lim_P f$ for the limit of $f$ on $P$. If we refer to [OSW86], and if $P$ is supposed to be a language, hence a r.e. set, a stabilized function on $\{P\}$ may be seen as a recursive set-driven learning function which converges on each text for $P$ (a text for $P$ is any infinite sequence which contains all and nothing but elements of $P$), and a stable point of the former as a locking-sequence of the latter. Let us suppose that we pay attention only to the limit on each element of $\mathcal{P}$; if every set in $\mathcal{P}$ is infinite, one can show that the notion of a stabilized function on $\mathcal{P}$ would be no more general if it were sensitive to the order in which the examples appear, or if it were undefined on a $y$ such that $D_y$ is not included in any element $P$ of $\mathcal{P}$.

We made the assumption that at each stage, the learner is provided with a finite set of evolutions, which is a stable point on the current set of rules of the stabilized function the learner implements. If we refer again to [OSW86] (i.e. if we think of a stable point as a locking-sequence), this assumption should be compared with the postulate, present in many paradigms, that every positive example will eventually appear: it implies that for a set-driven learning function and for any text, there is an initial segment of the text that constitutes a locking-sequence.

We can define our concept of a learning function, using the concept of a stabilized function; it associates each stage with the corresponding hypothesis, starting with an initial hypothesis $h_0$.

**Definition 3.5** *A learning function, on the learning problem $[c, r]$, from $h_0$, is determined by a stabilized function $f$ on $\{W_{\varphi_r(i)} \mid i \in \mathbb{N}\}$. It is denoted by $f^{[c,r,h_0]}$ and is defined on $\mathbb{N}$ by:*

*i)* $f^{[c,r,h_0]}(0) = h_0$;

*ii)* $f^{[c,r,h_0]}(n+1) = \lim_{W_{\varphi_r(f^{[c,r,h_0]}(n))}} f$.

We have not yet examined on what set of rules we might wish the learning procedure to end. It might be a coherent set of rules; we might want it to be total as well, or large

enough. We did propose a general framework for this matter. Here, we only need to give a special case of *progression*. Let us first explain with an example what this term is intended to catch, when dealing for instance with car driving. Let us suppose that at a given stage, the car will not leave the track as long as it is straight. A subgoal has been attained, and we might want it not to be lost at later stages. In that case, the controller of this stage reflects an ability acquired by the learner, and it is better than others. Of course, the car may crash in the next turn, and that is the reason why learning is not over; this controller is not the best. So relatively to one particular requirement, we need a family of preorders (reflexive and transitive binary relations) – which are defined on the class of all sets of rules –, one for each set of constraints $W_c$, and we write $a \preceq_c b$ if the relation relative to $W_c$ holds between the sets of rules $W_a$ and $W_b$; we call them *progression* relations.

Now, what shall we call a successful learning function? Let us assume that there is a best set of rules among all the sets of rules of the problem, relatively to the relation $\preceq_c$ (this hypothesis will be true in the following). First, we want the hypothesis of one stage to be 'greater or equal' (in the sense of $\preceq_c$) to the hypothesis of the previous one. Second, let us suppose that the learning function converges on a set of rules $R$; then $R$ must be maximal. The progression relation we will consider is the following: we define $a \preceq_c b$ to mean:

$$(W_a \cap W_c \neq \emptyset) \ \vee \ (\{e \in \mathbb{N} \mid (e \in W_a \setminus W_b) \wedge (\forall e' \in \mathbb{N})(e' \sqsubset e \rightarrow e' \in W_b)\} \text{ is finite}).$$

So when a learner has discovered a coherent set of rules, he cannot propose afterwards a set of rules that 'loses' an infinity of incomparable evolutions.

## 3.2 Learning and stages

Now, we define the concept of a learnable class of learning problems. The following definition is useful.

**Definition 3.6** *Let $\mathcal{S}$ be an r.e. class of learning problems (we denote the $j^{th}$ element of $\mathcal{S}$ by $[c_j, r_j]$). A* pre-paradigm *for $\mathcal{S}$ is defined by a progression relation and a set of recursive functions $f$ such that:*

$$(\forall j \in \mathbb{N})\ (\varphi_{f(j)} \textit{ is a stabilized function on } \{W_{\varphi_{r_j}(k)} \mid k \in \mathbb{N}\}).$$

Indeed, if a learning algorithm tries to learn all the problems of $\mathcal{S}$, such a function $f$ will exist.

**Definition 3.7** *Let an r.e. class $\mathcal{S}$ of learning problems, and a pre-paradigm $\mathcal{P}$ for $\mathcal{S}$ be given. We will say that $\mathcal{S}$ is* learnable *in $\mathcal{P}$ if there exists a function $f$ in $\mathcal{P}$ such that:*

$$(\forall i \in \mathbb{N})\ (\varphi_{f(j)}^{[c_j, r_j, 0]} \textit{ is a successful learning function}).$$

This means that each problem can be learned by a learner who starts in the same initial state (with the same controller). We supposed – without loss of generality — that learning starts with the controller first enumerated.

Now, we will consider particular classes of learning functions, and the corresponding pre-paradigms. Intuitively, these learning functions are defined by stabilized functions

that are looking for maximal values on a finite set of parameters, and the hypothesis is computed from these maximal values. This happens when the learning program tries to discover a qualitative model of the process. First, we must define the concept of a *description*. Intuitively, a description gives the number of parameters of a system, and the range of their potential values.

**Definition 3.8** *A* description *is a pair* $(\lambda, \rho)$, *where* $\lambda$ *is a recursive function, and* $\rho_{j,k}$, $j \in \mathbf{N}$, $1 \leq k \leq \lambda(j)$, *is a recursive enumeration of integers.*

**Definition 3.9** *Let* $\mathcal{S}$ *be an r.e. class of learning functions,* $(\lambda, \rho)$ *a description. A* pre-paradigm sup-$(\lambda, \rho)$ for $\mathcal{S}$ *is any pre-paradigm for* $\mathcal{S}$, *say* $\mathcal{P}$, *such that, for any function* $f$ *in* $\mathcal{P}$: *there exists a recursive function* $\lambda$, *there exists a recursive enumeration of integers* $\rho_{j,k}$, $j \in \mathbf{N}$, $1 \leq k \leq \lambda(j)$, *such that:*

> *there exists a recursive enumeration in* $j$ *of indices of recursive functions* $F_{j,1}, \ldots, F_{j,\lambda(j)}$ *and of a recursive function* $G_j$ *such that, for any integer* $n$:
>
> *i)* $(\forall k, 1 \leq k \leq \lambda(j))(\forall x \in \mathbf{N})\ (\mathbf{F_{j,k}(x)} \leq \rho_{\mathbf{j,k}})$ ;
>
> *ii)* $(\forall n \in \mathbf{N})(\varphi_{f(j)}(n) = G_j(\sup_{e \in D_n}(F_{j,1}(e)), \ldots, \sup_{e \in D_n}(F_{j,\lambda(j)}(e))))$.

We are now in a position to state the fundamental result [Mar93]:

**Proposition 3.1** *There exists an r.e. class* $\mathcal{S}$ *of learning problems, each containing only total sets of rules, among which there is a maximal one (with respect to the relation* $\preceq_c$ *where* $W_c$ *is the set of contraints of the problem), there exists a description* $(\lambda, \rho)$, *such that:* $\mathcal{S}$ *is learnable in the pre-paradigm sup-*$(\lambda, \rho)$ *for* $\mathcal{S}$, *and any learning function which learns* $\mathcal{S}$ *needs at least two stages in order to learn an infinite subclass of* $\mathcal{S}$.

# 4 Concluding remarks

In this paper we have tried to show how "learning to control a system" differs from other learning models and why we had thus to introduce a new theoretical framework. This framework relies on a few very general assumptions: the controllers constitute a recursively enumerable set of computable procedures and every controller chooses its action at each time among a finite set of possible actions. A learning program proceeds by stages: at the end of each stage, a new controller is proposed; this controller generates the data for the next stage which are analyzed and the program may then propose a different controller. In [Mar93] we have proved limiting results concerning the ability for a program or a class of learning programs to learn to control a class of systems, to converge or to progress ad infinitum, to learn in one stage or not, to take into account the outputs of the system or not...

Our learning model focuses on control and in order to be able to achieve non trivial results, we had to look for an adequate framework inside control theory: rule-based incremental control. This symbiosis between learning theory and control theory is one of the most original points of our approach.

Our theory lacks for now a notion of complexity, but it is our belief that we should first get better acquainted with the general concept of 'learnable', before we take on the notion of 'reasonably learnable'.

# References


[FKB+85] L. Foulloy, D. Kechemair, B. Burg, E. Lamotte, and B. Zavidovique. A rule based decision system for the robotization of metal laser cutting. In *IEEE Conf. on Robotics and Automation*, St Louis, MI, 1985.

[Fou90] L. Foulloy. *Du contrôle symbolique des processus: démarche, outils, exemples.* Thèse d'Etat, Université d'Orsay, Paris, 1990.

[Gol67] E.M. Gold. Language identification in the limit. *Information and Control*, 10, 1967.

[HLG91] D.A. Handelman, S.H. Lane, and J.J. Gelfand. Goal-directed encoding of task knowledge for robotic skill acquisition. In *IEEE International Symposium of Intelligent Control*, Arlington, VA, 1991.

[Luz91] D. Luzeaux. *Pour l'apprentissage, une nouvelle approche du contrôle: théorie et pratique.* Thèse de Troisième Cycle, Université d'Orsay, Paris, 1991.

[Luz92a] D. Luzeaux. How to deal with robot motion? Application to car-like robots. Technical report, University of California at Berkeley, No UCB-ERL-M92-7, 1992.

[Luz92b] D. Luzeaux. Miscellaneous on rule-based incremental control: from stability to learnability. Technical report, University of California at Berkeley, No UCB-ERL-M92-86, 1992.

[LZ90] D. Luzeaux and B. Zavidovique. Acquisition of a qualitative model. In *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Charleston, SC, 1990. ACM Press.

[Mar93] E. Martin. *L'apprentissage du contrôle sous contrôle récursif.* Thèse de Troisième Cycle, Paris VII, 1993.

[OSW86] D.N. Osherson, M. Stob, and S. Weinstein. *Systems that learn.* MIT Press, 1986.

[Rog67] H.Jr Rogers. *Theory of recursive functions and effective computability.* MacGraw-Hill, 1967.

[Val84] L.G. Valiant. A theory of learnable. *Communications of the ACM*, 27, 1984.