Claudio Sossai

Turing Machines, Uncertainty and Parallelism

AILA PREPRINT
n. 7 febbraio 1991

Direttore Responsabile: Ruggero Ferro
Iscrizione al Registro Stampa del Tribunale di Padova n. 1235 del 26.9.1990

Pubblicato con il contributo di:

Cassa di Risparmio di Padova e Rovigo

Stampa: Rotografica Padova

TURING MACHINES, UNCERTAINTY AND PARALLELISM

Claudio Sossai
Via Vlacovich 20 - 35100 Padova

Abstract.

A definition of machine is given which is an extension of Turing's definition and describes a proper extension of the concept of computability: we will show that for such machines the halting problem becomes decidable. The main characteristic of extended machines is the ability to work in situations of uncertainty and to carry on parallel computations. Parallelism and uncertainty appear in this frame as intertwined concepts.

1. Introduction.

In this paper we define a class of machines which is an extension of the class of Turing machines. A Turing machine in every step of the computation chooses between two possible instructions which must be executed depending on the value observed on the tape, which may be 0 or 1. The machines we will define are able to work with an extension of the integer numbers which, being the elements of a boolean valued model of arithmetic, have the following property: a number may be equal to more than one standard number with different degrees of truth. Thus the symbol which a machine may observe on the tape may be a number which at the same time is similar to 0 and 1; we then can say that the machine is in a situation of uncertainty. To decide which of the two possible instructions must be executed, the machine now has to decide whether the number 0 or 1 is represented, and to do this it needs a certain amount of information. For this reason, a new parameter is provided to any machine, expressing the amount of available information.

In this framework, three different possibilities, and hence three different kinds of computation, may arise:

1) The machine never finds situations of uncertainty. Then it works like a Turing machine.

2) The machine finds situations of uncertainty, but it has enough information to make a choice. In this case it makes some operations which are performable also by a Turing machine.

3) The machine has not enough information to decide a situation of uncertainty.

A standard Turing machine would have no chances to proceed in such a situation: here we propose two definitions of the concept of computation which allow the machine to go on. One is to conceive computations as a linear sequence of operations; then the machine will choose randomly between the two possible instructions and will extend its own information to the minimal piece of information which it should have had to make that same choice without uncertainty. According to our second definition, we allow the machine to proceed in parallel and execute both sequential computations corresponding to the two possible instructions; then the parallel computation can be represented as a tree, each branch of which represents one possible sequential computation and each branching is characterized by a piece of information. In either cases, all the operations which are performed by the machine with no change of information, are recursive. As we shall see, a machine with the sequential concept of computation does not correspond to a function. On the other hand, a parallel computation describes a function from extended integers into extended integers, and thus the computing ability of extended machines may be compared with that of Turing machines. We will use the halting problem as a test, and we will show that it becomes decidable in the extended sense.

I want to apologize for the presence of some concepts, for example that of random choice, which are undefined and must be taken as "primitive", and also for the style that is not in a definitive form, but I hope that the reader can overcome these difficulties and find some useful ideas.

1.1 Uncertainty.

The usual environment in which we can describe uncertainty is the Boolean algebra of possible events. The set of truth values $2 = \{0, 1\}$, where 0 stands for false and 1 for true, is a Boolean algebra; so it is possible to extend the classical semantics of logic to a semantics with uncertainty by interpreting the sentences in a Boolean algebra different from 2. There is an elegant theory where we can find all the tools we need: Boolean-valued models of set theory. An exposition of this theory may be found in [B]; here we give an informal description of the ideas contained in this theory that we shall need. A set X in the universe of sets V may be identified with its characteristic function c_X defined by:

$$c_X(x) = \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{if } x \notin X \end{cases}$$

So we can identify the universe of sets V with the class $V^{(2)}$ defined as follows (by recursion on the ordinal numbers α, β):

$$V_\alpha^{(2)} = \{f : \mathit{function}(f) \wedge (\exists\beta)[\beta < \alpha \wedge \mathit{domain}(f) \subseteq V_\beta^{(2)}] \wedge \mathit{range}(f) \subseteq 2\}$$

and

$$V^{(2)} = \{x : (\exists\alpha)(x \in V_\alpha^{(2)})\}.$$

Now if, in the previous definition, we replace 2 with a complete Boolean algebra B, it is possible to give a new definition that describes a new universe, named $V^{(B)}$, where we can interpret the sentences of set theory.

We can associate with every sentence ϕ of the first order language of set theory an element b of the algebra B which describes the truth value of the sentence and we shall indicate this with the formula $b = \|\phi\|$.

In the model $V^{(B)}$ logical axioms as well as all axioms of set theory have truth value 1_B and this property is preserved by the rules of inference.

1.2 Information.

Let B be a complete boolean algebra. A subset C of B is said to be dense in B iff $0 \notin C$ and for every element $b \in B$, with $b > 0$, there is an element $c \in C$ such that $c \leq b$. A dense subset of B is also called a set of information for B, and each $c \in C$ is called a piece of information.

A central idea in information theory is that if a is an event that has probability less than b, then the occurence of a gives us more information than the occurence of b. In our model of set theory we will not use measures and so we cannot speak of probability; however, it is well known that in a Boolean (sigma) algebra if $a \leq b$ then as soon as we define a measure μ necessarily $\mu(a) \leq \mu(b)$, so the partial order of the Boolean algebra captures some interesting properties of the idea of information.

Let c be a piece of information, i.e. an element of a dense subset of B, and let ϕ be a sentence; the well known forcing relation $c \Vdash \phi$ has among others some properties which capture the above idea.

In this frame, the forcing relation may be defined in the following way:

$$c \Vdash \phi \text{ iff } c \leq \|\phi\|.$$

Some of its properties can be interpreted as follows: $(d \leq c \wedge c \Vdash \phi) \rightarrow d \Vdash \phi$ captures the above idea of information theory, $(\forall c)(\exists d \leq c)(d \Vdash \phi \vee d \Vdash \neg\phi)$ says that any piece of information can be extended until the truth value of a sentence is decided, $(c \Vdash \phi) \rightarrow \neg(c \Vdash \neg\phi)$ shows the soundness of the definition.

Since we want to define a computable relation between information and sentences, first of all we need a complete Boolean algebra with a set of information in which Boolean operations are computable. Let G be 2^N, the set of all functions from the integers to 2. Let H be the set of functions from finite subsets of N in 2 partially ordered by inverse inclusion. For every $h \in H$ we put:

$$N(h) = \{f \in G : h \subset f\}.$$

Subsets of the form $N(h)$ form a base for the product topology on G when 2 is assigned the discrete topology. Each $N(h)$ is a clopen (i.e. closed and open) subset in this topology, in particular it is a regular open set, and it is easy to verify that the map $h \mapsto N(h)$ is an order isomorphism of H onto a dense subset of $RO(G)$, the set of the regular open elements of G. Therefore $\langle RO(G), N\rangle$ is a Boolean completion of H, and the latter is (up to isomorphism) a set of information for $RO(G)$. So if we take $B = RO(G)$ as a Boolean algebra, all the information we need is given us from the characteristic functions of the finite subsets of the integers, which are computable objects. We call C the isomorphic image of H into B and let A be the Boolean algebra generated by C. From now on when we will speak of the Boolean algebra B we will mean the complete boolean algebra $B = RO(G)$.

We will also use the following convention to simplify the notation: an element $h = \{\langle n_1, \gamma_1\rangle, \ldots \langle n_k, \gamma_k\rangle\}$, with $n_j \in N$ and $\gamma_j \in 2$ for every $j \leq k$, of H is denoted by $[n_1, \ldots, n_k]$, where each n_j is barred if and only if $\gamma_j = 0$. For instance, if $h \in H$ is defined as:

$$dom(h) = \{1, 4, 5\}$$
$$h(1) = 0, h(4) = 1, h(5) = 1,$$

then we will denote it by $[\overline{1}, 4, 5]$. Identifying h with its isomorphic image $N(h)$ in B, with this notation it is easy to see that for instance:

$$[1, 2, \overline{6}] \wedge [\overline{3}, 8] = [1, 2, \overline{3}, \overline{6}, 8]$$
$$\neg[1] = [\overline{1}]$$
$$\neg[1, 3] = [\overline{1}, \overline{3}] \vee [\overline{1}, 3] \vee [1, \overline{3}].$$

In this framework we can thus express two important concepts: uncertainty and information.

1.3 An arithmetic with uncertainty.

Recursive functions, Turing machines and computability have their natural environment in the natural numbers, and it is known that the set of finite ordinals is a model of the integers in set theory; so we can describe an arithmetic with uncertainty using the set of finite ordinals in a boolean valued extension of set theory.

What are such integers with uncertainty? It is known, that it is possible to define integers in set theory with a restricted formula $INT(x)$ and so integers in our model, that we shall call $N^{(B)}$, are all the elements x of $V^{(B)}$ such that $\|INT(x)\| = 1$. It is also possible to construct them explicitly.

First of all we can describe a copy of the integers in our model in this way: given an integer number n we have a copy of n, named $\hat{n}$, defined by:

$$\hat{n} = \{\langle \hat{j}, 1\rangle : j \in n\}.$$

The set of such numbers behaves just like the natural numbers and so from now on we shall identify $\hat{n}$ with n.

Recall that a partition of unit is a family $\{a_i\}_{i\in I}$ of elements of B such that:

(1) $a_i \wedge a_j = 0$ for every $i, j \in I$,
(2) $\bigvee_{i\in I} a_i = 1$.

If we have a partition of unit $\{a_i\}_{i\in I}$ and a family of integers $\{u_i\}_{i\in I}$, we can define a new element u, called mixture and written $u = \mathbf{\Sigma}_{i\in I} a_i \cdot u_i$, in the following way:

$$dom(u) = \bigcup_{i\in I} dom(u_i)$$
$$u(z) = \bigvee_{i\in I} [a_i \wedge \|z \in u\|] \text{ for } z \in dom(u).$$

We will usually write $u = b \cdot 1 \oplus \neg b \cdot 0$ for $u = \mathbf{\Sigma}_{i\in I} a_i \cdot n_i$ when $I = \{1, 2\}$, $a_1 = b$ and $a_2 = \neg b$.

It is possible to show that all the extended integers are mixtures of standard integers and that they enjoy all the first-order properties of standard integers. For example, we can see how the sum works in this extended arithmetic. We can follow the traditional way to define the sum, i.e. for $u \in N^{(B)}$ we let $u' = u \cup \{u\}$ and put $u + 0 = u$ and $u + v' = (u + v)'$.
We leave it to the reader to verify that with such definitions we have:

THEOREM 1. *If $\mathbf{\Sigma}_{i\in I} a_i \cdot n_i$ and $\mathbf{\Sigma}_{j\in J} b_j \cdot m_j$ are two elements of $N^{(B)}$ then*

$$\|\mathbf{\Sigma}_{i\in I} a_i \cdot n_i + \mathbf{\Sigma}_{j\in J} b_j \cdot m_j = \mathbf{\Sigma}_{i\in I, j\in J} (a_i \wedge b_j) \cdot (n_i + m_j)\| = 1$$

Note that in the following we will use the symbol $\mathbf{\Sigma}$ for mixtures, while $\sum$ indicates the sum.

COROLLARY 2. *Let $\mathbf{\Sigma}_{i\in I} a_i \cdot n_i$ be any element of $N^{(B)}$ where $I = \{0, \ldots, k\}$. Then there exist j_i and b_i, $i \leq k$, such that:*

$$\mathbf{\Sigma}_{i\in I} a_i \cdot n_i = \sum_{i=0}^{k} (\sum_{l=1}^{j_i} (b_i \cdot 1 \oplus \neg b_i \cdot 0))$$

In the following we will use only extended integers that are mixtures made by finite partitions of unit of elements of A.

This explains why in the sequel we can restrict to extended integers that are sums of numbers of the form $\neg c_i \cdot 0 \oplus c_i \cdot 1$.

2. Extended Turing machines.

We are now able to define Extended Turing machines, shortly ET machines. Remember that A is the Boolean algebra generated by the set of information C. To see the relation with standard Turing machines, it is convenient to recall the definition of Turing machine, as given e.g. by Rogers ([R], p.13). Let:

$Q = \{q_i : i \in N\}$ be the set of internal states,
$2 = \{0, 1\}$ be the set of scanned symbols,
$S = \{R, L\} \cup 2$ be the set of operations performed by the machine (where R means go to the right, L go to the left, 1 write 1 and 0 write 0).

DEFINITION 1. *A Turing machine t is a mapping from a finite subset of $Q \times 2$ into $S \times Q$.*

According to our philosophy, we first substitute 2 with the Boolean algebra A; so now A will be the set of possible symbols on the tape, and hence $S' = \{R, L\} \cup A$ is the set of operations which can be performed by the machine.

DEFINITION 2. *An Extended Turing machine τ is a mapping from a finite subset of $Q \times A$ into $S' \times Q$ that satisfies the following restriction: for any n, if $\langle q_n, b_1\rangle, \ldots, \langle q_n, b_m\rangle \in dom(\tau)$ are all pairs with initial state q_n, then it must be that $m = 2$ and $b_2 = \neg b_1$.*

As usual, if $\langle\langle q_i, b\rangle\rangle, \langle s, q_j\rangle\rangle \in \tau$, we say that $q_i b s q_j$ is an instruction of τ. Then the above restriction says that whenever $q_i b s q_j$ is an instruction of τ, then necessarily τ contains also an instruction beginning with $q_i \neg b$, and no other instruction with initial state q_i; we shall call macro instruction a pair of instructions of the form: $q_i b \ldots, q_i \neg b \ldots$.

2.1 Sequential computation with ET machines.

ET machines work with a tape which, like for standard Turing machines, has an infinite number of cells. However each cell may contain an element b of the Boolean algebra A. Such b is interpreted by the machine as the extended integer number $b \cdot 1 \oplus \neg b \cdot 0$. Note that when b is equal to $1 \in A$, then the number it represents is the number 1, while when $b = 0$, it is the number 0. This means that, if in this context we restrict to the boolean values 0 and 1 only, the tape will represent numbers formally as in the standard Turing machines.

Unlike Turing machines, ET machines have an additional memory where they can store the information achieved during the computation in the way we will specify below.

We can suppose that ET machines have a finite alphabet made of: $0, 1, [,], -$.

Then, for example, the element $[\bar{2}, 3]$ may be represented on the tape by:

$$\ldots 0\,|\,[\,|\,-\,|\,1\,|\,1\,|\,0\,|\,1\,|\,1\,|\,1\,|\,]\,|\,0\,|\ldots$$

The machine in every moment of the computation will be positioned on a cell which is called scanned symbol or *scs*.

In every moment of the computation the machine is in a certain internal state, q_i, which means it is pointed to the macro instruction $q_i b \ldots, q_i \neg b \ldots$. Then let a_1, a_2 be defined by:

$$a_1 = \|b \cdot 1 \oplus \neg b \cdot 0 = scs \cdot 1 \oplus \neg scs \cdot 0\|$$
$$a_2 = \|\neg b \cdot 1 \oplus b \cdot 0 = scs \cdot 1 \oplus \neg scs \cdot 0\|$$

We can say that a_1 represents the degree of likeliness between the number (represented by) b and the number (represented by) the scanned symbol; similarly, a_2 is the degree of likeliness between $\neg b$ and the scanned symbol.

First of all it is useful to see that $a_2 = \neg a_1$.

LEMMA 1. *If $a, b \in A$, then $\|a \cdot 1 \oplus \neg a \cdot 0 = b \cdot 1 \oplus \neg b \cdot 0\| = a \leftrightarrow b$.*

PROOF: To simplify notation, we call $n(b)$ the number represented by b on the tape, i.e. $n(b) = b \cdot 1 \oplus \neg b \cdot 0$. It is easy to see that $dom(n(a)) = dom(n(b)) = \{\emptyset\}$. Applying the definition of equality we have:

$$\begin{aligned}\|n(a) = n(b)\| &= (n(a)(\emptyset) \to \|\emptyset \in n(b)\|) \wedge (n(b)(\emptyset) \to \|\emptyset \in n(a)\|) = \\ &= (((a \wedge \|\emptyset \in 1\|) \vee (\neg a \wedge \|\emptyset \in 0\|)) \to \|\emptyset \in n(b)\|) \\ &\quad \wedge (((b \wedge \|\emptyset \in 1\|) \vee (\neg b \wedge \|\emptyset \in 0\|)) \to \|\emptyset \in n(a)\|) = \\ &= (((a \wedge 1) \vee (\neg a \wedge 0)) \to b) \wedge (((b \wedge 1) \vee (\neg b \wedge 0)) \to a) = \\ &= a \leftrightarrow b. \diamondsuit\end{aligned}$$

PROPOSITION 2. *If a_1 and a_2 are defined as above,then $a_2 = \neg a_1$.*

PROOF: Using the previous lemma we have $a_1 = b \leftrightarrow scs$ and $a_2 = \neg b \leftrightarrow scs$ and using the tautologies $((a \leftrightarrow b) \vee (\neg a \leftrightarrow b))$ and $\neg((a \leftrightarrow b) \wedge (\neg a \leftrightarrow b))$ we have that $a_1 \vee a_2 = 1$, $a_1 \wedge a_2 = 0$ and therefore $a_2 = \neg a_1$. $\diamondsuit$

We assume that the machine can start the computation only if it is provided with a piece of information consisting of an element c in C. This element may vary in the course of computation, as we will now explain; we will call it the information available to the machine in that moment.

If at a certain moment the machine is in state q_i, the values a_1, a_2 are defined as above and c is the information available in that moment, then only three cases are possible and mutually exclusive:

(1) $c \leq a_1$
(2) $c \leq a_2$, that is $c \leq \neg a_1$
(3) $c \not\leq a_1$ and $c \not\leq a_2$.

If the first case occurs, we have from the definition of forcing that

$$c \Vdash b \cdot 1 \oplus \neg b \cdot 0 = scs \cdot 1 \oplus \neg scs \cdot 0. \tag{1}$$

Then we may say that the information c available to the machine is enough to force equality between the number represented by the scanned symbol and that represented in the first instruction $q_i b \ldots$, and hence the machine will choose and execute this instruction; similarly if $c \leq a_2$ holds, it means that

$$c \Vdash \neg b \cdot 1 \oplus b \cdot 0 = scs \cdot 1 \oplus \neg scs \cdot 0 \tag{2}$$

and so the machine will execute the instruction $q_i \neg b \ldots$

If neither the first nor the second case occurs, we may say that the information c available to the machine is not enough to decide whether the number represented by the scanned symbol is equal to the number represented in the first instruction or the number in the second instruction. We then put $c_1 = c \wedge a_1$ and $c_2 = c \wedge a_2$; obviously, $c_1, c_2 \geq 0$ and $c_1 \Vdash b \cdot 1 \oplus \neg b \cdot 0 = scs \cdot 1 \oplus \neg scs \cdot 0$ and $c_2 \Vdash \neg b \cdot 1 \oplus b \cdot 0 = scs \cdot 1 \oplus \neg scs \cdot 0$. In this case the machine stops and waits until an external agent, for example a generator of random numbers, provides it with a number, 0 or 1, which the machine takes as indication whether the first or the second instruction must be executed, respectively. It is important to note that the external agent has no knowledge about the machine and its computation; thus we could say that the machine chooses randomly. Moreover, if 0 is provided, the machine extends its information from c to c_1, and similarly, if 1 is provided, it extends its information from c to c_2; in other words, after a random choice, the machine extends its information in such a way that, if it would have had that information before, it would have executed the same instruction without any indication from the external agent. Even more informally, the machines assumes to have a posteriori the information which justifies the choice which it actually has taken randomly[1].

So if it chooses to extend the information c with c_1, it will execute the first instruction, and the second otherwise. It is worthwhile to note that in this way to every random choice there corresponds a proper extension of the information previously available to the machine. It is easy to see that in the case of the Turing machine, namely when if $b = 1$ or $b = 0$ and $scs = 1$ or $scs = 0$, no information is needed to make a choice.

Once the machine has chosen the instruction, it executes the operation indicated by the instruction, that is: if the operation is L then it moves the tape one cell left, if the operation is R then it moves the tape one cell right and if the operation is b then it writes b in the scanned cell.

[1] After some conversations on this point, G. Sambin proposed to refer to such a behaviour of ET machines as "The Jesuit's principle".

The halting condition occurs when the machine is in a state to which there corresponds no macro instruction.

Since the computation may depend on random choices, it is quite impossible to determine the behaviour of the machine, and so the concept of function does not correspond to this kind of computation. In section 3 we will describe another kind of computation for ET machine, one that is specifically devised to capture the concept of function.

It is possible to describe (see [R], p.15) every computation made by a Turing machine t provided with input n, with a sequence of pairs $\langle q_i, o \rangle$, called instantaneous descriptions, formed by an internal state q_i and a state of the tape o (which includes the indication of the scanned symbol). We have seen that the actual value of the symbol scanned by an ET machine sometimes is not in general sufficient for the machine to choose between the two parts of a macro instruction; however, if we know also the information c reached by the machine after the execution of the instruction, then we can say which of the two instructions has been executed. Thus we can give the following definition.

DEFINITION 3. *An instantaneous description for ET machines is a triple $\langle q_i, o, c \rangle$, where q_i and o are an internal state and a state of the tape, and c is the information reached by the machine after the execution of the macro instruction corresponding to q_i.*

We suppose that a machine employs the same amount of time to execute any instruction, and we call the moment of the execution of an instruction a step of the computation.

Since every instantaneous description is generated by the execution of an instruction, we can index the set of instantaneous descriptions by the corresponding step number.

We will indicate with $T(\tau(y))$ the sequence of instantaneous descriptions, generated by the machine τ with input y and indexed by step numbers.

2.2 First analysis of ET sequential computations.

Since the set A of symbols is recursively enumerable, the totality of ET machines can be equipped with à Gödel numbering. At every instant of a computation, only a finite number of cells contains a nonzero symbol. Accordingly, we call state of the tape the description of this part with the indication of the scanned cell. It is possible to give a coding for the possible states in which the tape can be. From now on we shall identify possible states of the tape and Extended Turing Machines with their codings.

We have noted that the set of programs of the Turing machines is a subset of the set of all programs in the extended sense. But if we take a program of a Turing machine, it behaves like a Turing machine only if the input is a standard integer. In fact, if the input is an extended integer it is possible that

a program of a Turing machine reaches a condition of uncertainty and so it must make a random choice. So the programs of Turing machines, from the viewpoint of extended machines, represent Turing machines only for standard inputs.

If the machine x with input y stops and gives output z we call $\langle y, z\rangle$ an input output pair.

DEFINITION 1. *We call graph of the machine x the set $G(x)$ of all pairs of input output of the machine x started with no information (i.e. with 1 as information).*

THEOREM 2. *There exists an ET machine τ whose graph consists only of standard integers, but for no Turing machine t, $G(\tau) = G(t)$.*

PROOF: It is easy to see that there is a machine which decides if a number is a standard integer or not. So we can define the machine τ as follows: for any input y if y is not a standard integer then τ with y as input loops, otherwise it writes 1 on the *scs* and passes the control to the following macro instruction:

$$q_i[0]1q_{i+1}$$
$$q_i\neg[0]1q_i$$
$$q_{i+1}stop$$

It is easy to see that τ with input y gives no output if y is not a standard integer, otherwise it stops or not depending on a random choice. So no Turing machine can have the same graph since it is not r.e. $\diamondsuit$

DEFINITION 3. *We define a predicate S in the following way: $S(x, y, c, p, z, d)$ holds iff the machine x, with input y, and starting information c, in p steps reaches the tape state z with terminal information d and then stops.*

Note that if no random choiches are made during the computation then all the operations performed are combinations of boolean operations over A, which we know to be recursive operations, and that at the end of the computation, the machine has the information sufficient to make all choices needed to end the program. So if we start again the machine, then it will repeat the same computation, but now with no random choices. All these observations are summarized in the following:

PROPOSITION 4. *If $S(x, y, c, p, z, d)$ and $c = d$, then the operations performed to compute z are recursive operations.*

PROPOSITION 5. *For all x, y, c, if there are p, z, d such that $S(x, y, c, p, z, d)$ then we have $S(x, y, d, p, z, d)$.*

In information theory the amount of information contained in an element b of the algebra of possible events is defined as $I(b) = -\log(\mu(b))$, where μ is a probability measure on the algebra of possible events.

THEOREM 6. *For every probabilistic measure μ on B, if $S(x, y, c, p, z, d)$ and $c \neq d$ then $I(c) < I(d)$.*

PROOF: We have already noticed that if $c \neq d$ then $c > d$. Any measure μ is monotonic, and then $\mu(c) > \mu(d)$, so that $I(c) < I(d)$. $\diamondsuit$

Roughly speaking, the theorem states that after every random choice, the amount of information available to the machine increases.

3. Parallel computations with ET machines.

If we start two copies of the same ET machines, even with the same input and information, it may well happen that they follow two different sequential computations and give two different computations. This is due to the dependence of sequential computations on random choices.

For this reason, ET machine with the sequential concept of computation do not correspond to our intuitive idea of "computing a function"; thus, if we want to compare ET with Turing machines, we must give a more mathematical description of the concept of computation.

Randomness, in the description we give, appears like a primitive and undefined concept. One way to analyze its role and meaning in ET machines, may be found by answering the question: what must we do formally, so that all the power of the machine remains but randomness disappears? The solution of the problem lies in passing from a sequential computing mechanism to a parallel one. Let us see how. Randomness comes into play when the information c available to the machine is not enough to choose between two instructions. This means that if $q_ib\ldots q_i\neg b\ldots$ are the two instructions, $a_1 = \|b\cdot 1\oplus\neg b\cdot 0 = scs\cdot 1\oplus\neg scs\cdot 0\|$ and $a_2 = \|\neg b\cdot 1\oplus b\cdot 0 = scs\cdot 1\oplus\neg scs\cdot 0\|$, where *scs* is the value of the scanned symbol, then $c \not\leq a_1$ and $c \not\leq a_2$. The role of randomness was to choose between the two possible extensions of information, $c\wedge a_1 = c_1$ and $c\wedge a_2 = c_2$, and thus between the two instructions. Now, if we could parallely compute with the same machine but with two different information c_1 and c_2, no random choice would be necessary and so randomness would be eliminated. This is precisely what we want to do in describing parallel computations with ET machines.

To describe parallel computations we must add the following new ability to ET machines: an ET machine τ is able in every moment of the computation, if necessary, to produce a copy of its program, a copy of the tape and to carry on a new parallel computation with the new tape and a new state of information. So when the information c possessed by the machine is not enough to decide which instruction must be executed, the machine creates a copy of itself and carries on two computations, one with the new information c_1 and the other with the new information c_2. In this way random choices completely disappear from the computation of ET machines.

We have supposed, in ET sequential computations, that the same time is needed to execute each instruction and we called the moment of the execution

of an instruction a step of the computation. Similarly, we suppose that in a parallel computation the same time is needed to execute each instruction. We will also suppose that no time is required to an ET machine to duplicate its program and the tape, so that, also if there are duplications of the computation, the execution of the instructions remains synchronous between parallel branches.

Accordingly we can call step of a parallel computation the moment of the simultaneous execution of all instructions that are performed contemporarily in all the branches of the computation. In the parallel case, the set $T(\tau(y))$ of all instantaneous descriptions, indexed by the corresponding step number, of the parallel computation of $\tau(y)$ is not a linear sequence but rather a possibly infinite binary tree. A branch of the tree $T(\tau(y))$ is a sequence of instantaneous descriptions corresponding to a sequential computation of $\tau(y)$.

Let c be a piece of information contained in an instantaneous description of $T(\tau(y))$; we will denote by $\tau^c(y)$ the computation τ with input y started with information c. Note that $T(\tau^c(y))$ is the subtree of $T(\tau(y))$ determined as follows. Assume that the first step in which c appears is p, and that $\langle q_m, o_m, c\rangle$ is the corresponding instantaneous description. Then an instantaneous description $\langle q, o, d\rangle \in T(\tau(y))$ belongs to $T(\tau^c(y))$ if either it corresponds to a step n with $n \leq p$ and $d \geq c$ or it corresponds to a step l with $l \geq p$ and $d \leq c$.

DEFINITION 1. *We will call states of information of a step p of the computation $\tau(y)$, all the pieces of information that belong to some instantaneous description of $T(\tau(y))$ with index p.*

DEFINITION 2. *$T(\tau^c(y))$ is a terminal branch in the computation $\tau(y)$ if $T(\tau^c(y))$ is the set of all instantaneous descriptions generated by a sequential computation that stops with an instantaneous description where c appears.*

This means that the computation $\tau^c(y)$ is a sequential computation that stops with no random choices.

DEFINITION 3. *A state of information c of a step p, is an active state if c does not determine a terminal branch at step p.*

If a subtree of the computation is a terminal branch, we suppose that, after it has reached the stop condition, in every unitary interval of time it makes a copy of its tape. The reason is, as we shall now see, that in every moment of the computation we must know the state of the tape of every branch, and it is easier to produce a copy of the tape of the terminal branches, than to search, in every moment, for the tapes of the branches that have stopped.

3.1 Output of a parallel computation.

We have noticed that when a machine changes its state of information c into two new states c_1 and c_2, then c_1 and c_2 satisfy: $c_1 \vee c_2 = c$ and

$c_1 \wedge c_2 = 0$. Therefore, since an ET machine always starts with information 1, in every step of the computation, if $\{c_1, \ldots, c_n\}$ are the states of information corresponding to that step, we have that: $c_1 \vee \cdots \vee c_n = 1$ and $c_h \wedge c_k = 0$ for $h, k \leq n$ and $h \neq k$; this means that $\{c_1, \ldots, c_n\}$ is a partition of unity.

Every step p of the computation, uniquely determines two finite sets: the set $\{u_1, \ldots, u_n\}$ of all numbers represented on all the tapes and the set $\{c_1, \ldots, c_n\}$ of the corresponding states of information.

For the reason that $u_1, \ldots, u_n$ represent extended integers and $\{c_1, \ldots, c_n\}$ is a partition of unit, we have that $u = \Sigma_{i \leq n} c_i \cdot u_i$ is a new extended integer.

DEFINITION 1. *We say that the number $u = \Sigma_{i \leq n} c_i \cdot u_i$ is the number represented on the tape(s) at that step of the computation, and we will write $\tau_p(y) = u$.*

Note that for an ET machine it may happen that, for some input, there are some branches which are terminal branches and some other branches which do not stop. This may be a problem if we want to define the output of an ET machine.

We can overpass this problem simply considering a branch that never changes the number represented on its tape, as a branch where the computation is completed; and so we can give the following definition of the output of an ET machine τ with input y.

DEFINITION 2. *The output of an ET machine τ with input y is z, written $\tau(y) = z$, if there exists p such that $\|\tau_p(y) = z\| = 1$, and for every $q \geq p$, we have $\|\tau_q(y) = z\| = 1$.*

Note that $\tau(y) = z$ may be not Turing decidable, and we can say that, in general, an ET machine, in a finite number of steps, can give us only a partial information about its output, for example: the output of $\tau(y)$ would be surely z if the machine could have the information c. Anyway this is not a problem for ET machines, because we have constructed them so that their main characteristic is the ability to work in conditions of partial information.

3.2 Sequential and parallel composition of ET machines.

We have seen that for ET machines there are two ways of computing: the sequential way and the parallel way. Similarly, there will by a sequential mode of composing machines and a parallel one.

DEFINITION 1. *Let σ, τ be two ET machines and y be an extended integer; the sequential composition of σ, τ , written $\sigma(\tau)$, is the new machine which applied to y behaves in the following way:*

1) starts the computation $\tau(y)$,

2) if and when the parallel computation $\tau(y)$ meets a stop condition with terminal branch $T(\tau^c(y))$ and corresponding tape number u, then the computation $\sigma^c(u)$ is started.

Note that, if at some step p, $\tau(y)$ meets more than one stop condition with terminal branches determined by $c_1, \ldots, c_n$, and corresponding tape numbers $u_1, \ldots, u_n$, then simultaneously all the computations $\sigma^{c_1}(u_1), \ldots, \sigma^{c_n}(u_n)$ start.

Now let us say what is the number represented at step p on the tapes of the computation $\sigma(\tau(y))$. Suppose that at step p the machine $\sigma(\tau(y))$ is in the following situation:

1) $c_1, \ldots, c_n$ are the active states of information of $\tau(y)$, with numbers $u_1, \ldots, u_n$ represented in the corresponding tapes, and $c_{n+1}, \ldots c_{n+m}$ are the states of information corresponding to terminated branches, with corresponding numbers $u_{n+1}, \ldots, u_{n+m}$,

2) $b_1, \ldots, b_l$ are the states of information and corresponding tape numbers $w_1, \ldots, w_l$, produced by the computations $\sigma^{c_{n+1}}(u_{n+1}), \ldots, \sigma^{c_{n+m}}(u_{n+m})$ generated by the terminal branches determined by $c_{n+1}, \ldots, c_{n+m}$.

We have already seen that the family $\{c_j\}_{j \leq n+m}$ is a partition of unit; and it is easy to check that also the family $\{a_h\}_{h \leq n+l}$ where $a_i = c_i$ for $i \leq n$, and $a_{n+j} = b_j$ for $j \leq l$ is a partition of unit.

DEFINITION 2. *Let $\{v_h\}_{h \leq n+l}$ be the family of extended integers with the following property: $v_i = u_i$ for $i \leq n$, and $v_{n+j} = w_j$ for $j \leq l$; then the extended number $v = \Sigma_{i \leq n+l} a_i \cdot v_i$ is the number represented on the tapes at step p of the computation $\sigma(\tau(y))$.*

Intuitively, the number represented on the tapes of $\sigma(\tau(y))$, is the mixture generated by the active branches of $\tau(y)$ at step p together with all branches of the computations of σ started from terminated branches of τ.

Note that the result of a sequential composition of ET machines is an ET machine. In fact let the sequential composition be $\sigma(\tau(y))$, and let c determine a terminal branch of $\tau(y)$; this means that $\tau^c(y)$ ends with an instruction of the form: $q_i a s q_j$ but no instructions of the form q_j appears in τ. So we can add to τ the program obtained from σ by modifying in an obvious way the indexes of internal states.

This method of composing ET machines is very close to the usual one: if and when one machine stops, the other takes the output of the first machine as input and starts the computation; thus we shall now define a new method of composing machines: the notion of parallel composition of concurrently computing machines.

DEFINITION 3. *The parallel composition of two ET machines σ, τ, that we will indicate with the symbol $\sigma \otimes \tau$, is the procedure defined, for every pair of extended integers x, y, in the following way:*

1) start simultaneously the two computations $\sigma(x)$ and $\tau(y)$, so that in every moment the two machines have performed the same number of steps.

2) Let $\{a_i\}_{i \leq n}$ be the active states of information at step p of the computation $\sigma(x)$, and $\{v_i\}_{i \leq n}$ the numbers represented on the corresponding tapes.

If and when at step p the computation $\tau(y)$ reaches a stop condition with terminal branch determined by c, and value u on the corresponding tape, then every computation $\sigma^{a_i}(v_i)$, satisfying $0 < c \wedge a_i < a_i$ splits into the new computations $\sigma^{a_i \wedge c}(u)$ and $\sigma^{a_i \wedge \neg c}(v_i)$.

To clarify the above definition, it is simpler to assume $x = y$. Then note that by the condition $0 < c \wedge a_i$, $\tau^{c \wedge a_i}(x)$ is defined, and since obviously $(c \wedge a_i) \leq c$ we have that $\tau^{c \wedge a_i}(x) = u$. The meaning of the second part of the above condition, that is $(c \wedge a_i) < a_i$, is to make sure that $\sigma(x)$ is in that moment computing using a piece of information a_i, which may be split through c, and extended to two different hypotheses $a_i \wedge c$ and $a_i \wedge \neg c$. Since $a_i \wedge c$ and $a_i \wedge \neg c$ are proper extensions of a_i, $\sigma(x)$ knows nothing about $\sigma^{a_i \wedge c}(x)$ and $\sigma^{a_i \wedge \neg c}(x)$. On the contrary, $\tau(x)$ knows that $\tau^{c \wedge a_i}(x) = u$, and thus $\sigma(x)$ utilizes the output u reached by $\tau(x)$ to proceed with the computation $\sigma^{a_i \wedge c}(u)$, while it goes on with its own value v in the computation $\sigma^{a_i \wedge \neg c}(x)$.

The following assertions are easy consequences of the definition.

1) If there are at least two terminal branches at step p in the computation $\tau(y)$, the order in which they split the computation $\sigma(x)$ does not modify the computation itself. In fact, if for example $\tau(y)$ at step p reaches the terminal branches determined by c_1 and c_2 and corresponding numbers u_1 and u_2, then the final result is in any case that the computation of σ is split into the three computations $\sigma^{a_i \wedge c_1 \wedge \neg c_2}(u_1)$, $\sigma^{a_i \wedge \neg c_1 \wedge c_2}(u_2)$ and $\sigma^{a_i \wedge \neg c_1 \wedge \neg c_2}(v_i)$, since $a_i \wedge c_1 \wedge c_2 = 0$.

2) Imagine that the computation $\tau(y)$ reaches a terminal branch determined by c in the same moment in which the computation $\sigma(x)$ splits with information a_1 and a_2; then the same computation is obtained if we first split the computation $\sigma(x)$ with information a_1, a_2, and then each branch with $c, \neg c$, or conversely.

Moreover, note that the meaning of the condition $a_i \wedge c < a_i$ is also to avoid possible conflicts of opinion between σ and τ; in fact, if it is $a_i = c$, then there is no splitting and the computation of $\sigma^{a_i}(v_i)$ is not changed.

It is easy to generalize the parallel composition of two ET machines to a finite number of ET machines.

DEFINITION 4. *The parallel composition of $\sigma_1, ..., \sigma_m$, written $\sigma_1 \otimes ... \otimes \sigma_m$, is the behavior defined, for every extended integers $x_1, ..., x_m$, as follows:*

1) start simultaneously the computations $\sigma_1(x_1), ..., \sigma_m(x_m)$,

2) let $\{a_{j,i}\}_{i \leq n_j}$ be the active states of information at step p of the computation $\sigma_j(x_j)$, and $\{v_{j,i}\}_{i \leq n_j}$ the numbers represented on the corresponding tapes, for every $j \leq m$. If and when, for every $j < m$, at step p the computation $\sigma_{j+1}(x_{j+1})$ reaches a stop condition with terminal branch determined by c_{j+1}, and value u_{j+1} on the corresponding tape, then every computation $\sigma_j^{a_{j,i}}(v_{j,i})$ of $\sigma_j(x_j)$, satisfying $0 < c_{j+1} \wedge a_{j,i} < a_{j,i}$ splits into the new computations $\sigma_j^{a_{j,i} \wedge c_{j+1}}(u_{j+1})$ and $\sigma_j^{a_{j,i} \wedge \neg c_{j+1}}(v_{j,i})$.

Now we can define the value represented on the tapes of a parallel composition of ET machines.

DEFINITION 5. *Let $\sigma_1, ..., \sigma_n$ be ET machines, the value represented on the tapes of the computation $\sigma_1 \otimes ... \otimes \sigma_n$ at step p is the number represented on the tapes of σ_1 at step p.*

It is easy to see that the definition of output of $\sigma_1 \otimes ... \otimes \sigma_n$ is the same as the one we have given for an ET machine.

We will now describe a new kind of dynamic memory for ET machines.

Let ψ be an ET machine representing the identity function (e.g. the machine with the program $q_0 1 1 q_0, q_0 0 0 q_0$); then for every ET machine τ and numbers x, y, the machine $\psi(x)$ parallely composed with $\tau(y)$ behaves like a dynamic memory. In fact assume that c determines a terminal branch with tape number u at step p in the computation $\tau(y)$; then the tape of the branch c of the machine $\psi(x)$ in the computation $\psi(x) \otimes \tau(y)$ after the step p has the value u and never changes it.

Contrary to the case of sequential composition, it is still an open problem whether a parallel composition of ET machines is again an ET machine (even if my conjecture is that the answer is no).

We can observe that if $\sigma = \sigma_1 \otimes ... \otimes \sigma_n$ and $\tau = \tau_1 \otimes ... \otimes \tau_m$ are two parallel compositions of ET machines we can define their parallel composition as: $\sigma \otimes \tau = \sigma_1 \otimes ... \otimes \sigma_n \otimes \tau_1 \otimes ... \otimes \tau_m$. Thus the class formed by all ET machines and all the finite parallel compositions of ET machines is closed under parallel composition, i.e. the parallel composition of two elements of this class is an element of the class.

DEFINITION 6. *We call Cooperative Extended Turing machine, shortly CET machine, a finite parallel combination of ET machine.*

4. Computability of the halting problem for CET machines.

Given a CET machine τ with input y we are also interested in the set of all pieces of information c that characterize terminating branches. We can describe this set as: $\eta(\tau(y)) = \{c : c \text{ determines a terminal branch}\}$, in fact for every information $d \leq \bigvee \eta(\tau(y))$ we have that $\tau^d(y)$ surely stops.

We are now able to compare the computing ability of Turing machines with the one of CET machines, and we will use the halting problem to investigate the difference between them in facing this task.

We have noticed that a CET machine τ with input y may have some branches that are terminal branches and some others where the computation never reaches a stop condition; for this reason the halting problem must be redefined in this framework.

First we will extend to CET machines the familiar notion $\phi_x(x)\downarrow$, meaning that the Turing machine ϕ_x with input x stops. We shall then give a description of the truth degree b with which a CET machine τ with input y can

reach a terminating branch.

By the above remarks, it is reasonable to define this value as $\|\tau(y)\downarrow\| = \bigvee \eta(\tau(y))$; and then the degree of truth that a CET machine τ with input y can not reach a terminal branch, may be defined as $\|\tau(y)\uparrow\| = \neg\|\tau(y)\downarrow\|$.

For Turing machines the halting problem may be formulated as the task of calculating the characteristic function of the set: $K = \{x : \phi_x(x)\downarrow\}$. The characteristic function of K may be described as $c_K(x) = \|\phi_x(x)\downarrow\|$, where for Turing machines the only possible values for $\|\phi_x(x)\downarrow\|$ are 0 and 1.

We have seen that $\|\phi_x(x)\downarrow\|$, when x is seen as the code for a CET machine, may have a value $0 < b < 1$ and so it is quite reasonable to expect that for CET machines K becomes a Boolean valued set. For this reason, if we call $K^{(B)}$ the set corresponding to K for CET machines, then it is meaningful to require that its characteristic function has the following properties:

$$\|c_{K^{(B)}}(x) = 1\| = \|x \in K^{(B)}\| = \|\phi_x(x)\downarrow\|$$
$$\|c_{K^{(B)}}(x) = 0\| = \|x \notin K^{(B)}\| = \|\phi_x(x)\uparrow\|$$

But all such properties together characterize uniquely the following function:

$$c_{K^{(B)}}(x) = \|\phi_x(x)\downarrow\| \cdot 1 \oplus \|\phi_x(x)\uparrow\| \cdot 0$$

which we will consider as the characteristic function of $K^{(B)}$ in the CET sense.

Before showing that there is a CET machine that computes the above function, we must prove the following property of the stop sets.

THEOREM 1. *For every $y \in N^{(B)}$ with finite domain, and $x \in N$, the set $\eta(\phi_x(y))$ is finite.*

PROOF: We will prove the theorem by induction on the number of parallel compositions used to define ϕ_x.

Assume that no parallel compositions appear in ϕ_x, i.e. ϕ_x is a single ET machine. We have seen in paragraph 1.3 that all extended numbers $y \in N^{(B)}$ that we use have finite a domain, and so y may be described as a finite sum of numbers of the form $a_i \cdot 1 \oplus \neg a_i \cdot 0$.

Moreover, the program ϕ_x is made of a finite number h of instructions of the form $q_i b s q_j$, and to simplify we shall call b the left symbol of the instruction and s the right one. Let S_1 be the set of all left symbols of the program ϕ_x, while an element of S_2 is either a right symbol different from R, L, or an element a_i that appears in the decomposition of y. Obviously S_1 and S_2 are finite sets. We then put:

$$Z = \{\bigwedge_{i \leq k} (b_i \leftrightarrow c_i) : \text{ for some } k \in N, b_1, \ldots, b_k \in S_1, c_1, \ldots c_k \in S_2\}.$$

It is clear that Z is a finite set too. Hence it is enough to show that $\eta(\phi_x(y))$ is contained in Z. Assume $e \in \eta(\phi_x(y))$; we show that $e \in Z$ by induction on the number l of extensions of initial information that were made to reach e. Note that, since $e \in \eta(\phi_x(y))$, l is surely finite.

If $l = 1$, than $e = 1 \wedge a_1$ or $e = 1 \wedge a_2$, where $a_1 = \|b \cdot 1 \oplus \neg b \cdot 0 = scs \cdot 1 \oplus \neg scs \cdot 0\|$ and $a_2 = \|\neg b \cdot 1 \oplus b \cdot 0 = scs \cdot 1 \oplus \neg scs \cdot 0\|$ for some $b, \neg b \in S_1$. By §2.1 , Lemma 1, this means that $a_1 = b \leftrightarrow scs$ and $a_2 = \neg b \leftrightarrow scs$, for some $b, \neg b \in S_1$. It is easy to see that $scs \in S_2$ because scs may be only either a number a_i already written on the tape when the machine starts, or a number written by the machine during the computation, i.e. a right symbol; in any case e has the form $b \leftrightarrow c$ and thus it is an element of Z.

Let $l = m + 1$, i.e. $e = d \wedge a_1$ or $e = d \wedge a_2$, where $d \in Z$ by the inductive hypothesis, and a_1 and a_2 have the same form as in the previous step of the proof, i.e. $a_1 = b_1 \leftrightarrow c_1$, and $a_2 = b_2 \leftrightarrow c_2$ where b_1, b_2 are left symbols and c_1, c_2 are right symbols. Thus $e = d \wedge (b_1 \leftrightarrow c_1)$ or $e = d \wedge (b_2 \leftrightarrow c_2)$, where $d \in Z$, and hence $e \in Z$. And this concludes the proof for the case of a single ET machine.

Now assume $\phi_x(y) = \sigma(y_1) \otimes \tau(y_2)$ where $y = \langle y_1, y_2 \rangle$.

By inductive hypothesis we may assume that $\eta(\sigma(y_1)) = \{c_1, \ldots, c_n\}$, and $\eta(\tau(y_2)) = \{d_1, \ldots, d_m\}$ with corresponding numbers $\{u_1, \ldots, u_m\}$ represented on the tapes. But then also $\eta(\sigma^{d_i}(u_i))$ is finite for every $i \leq m$, and thus also $\eta(\sigma(y_1) \otimes \tau(y_2))$ is finite, since it is contained in $\bigcup_{i \leq m} \eta(\sigma^{d_i}(u_i)) \cup \{c_1, \ldots, c_n\}$. $\diamondsuit$

Now we are in a position to prove:

THEOREM 2. *The set $K^{(B)}$ is a CET computable set.*

PROOF: What we must do is to construct a CET machine τ such that for every $n \in N$ we have $\|\tau(n) = c_{K^{(B)}}(n)\| = 1$, where $c_{K^{(B)}}$ is the characteristic function of $K^{(B)}$ that we described.

We can construct τ by composing two ET machines with the dynamic memory. For every $n \in N$, the first machine ρ is the machine which builds up the ϕ_n machine and starts the computation $\phi_n(n)$. The second machine σ is the machine which for every extended integer u as input gives the number 1 as output.

We then define τ to be obtained by sequential composition of ρ and σ, followed by parallel composition with ψ, i.e. we put $\tau(n) = \pi(0, n) = \psi(0) \otimes \sigma(\rho(n))$.

We now show that $\|(\tau(n) = c_{K^{(B)}}(n))\| = 1$.

In fact, by the preceding theorem the stop set of $\sigma(\rho(n))$ is finite; if it is empty, then $\sigma(\rho(n))$ diverges on any branch, then the tape of ψ is never changed, and hence the output of τ is 0.

Now assume $\eta(\sigma(\rho(n)))$ is $\{c_1, \ldots, c_n\}$, and let $p_1, \ldots, p_n$ be the steps necessary to reach the respective stop condition.

At step p_1, $\phi_n(n)$ reaches a stop condition with information c_1 and tape number u_1, the computation $\sigma^{c_1}(u_1) = 1$ causes the tape of $\psi(0)$ to be split into two tapes: one with tape number 1 and corresponding information c_1, the other with tape number 0 and corresponding information $\neg c_1$.

If and when at step p_2, $\phi_n(n)$ reaches the second stop condition with information c_2 and tape number u_2, the computation $\sigma^{c_2}(u_1) = 1$ causes the tape of $\psi^{\neg c_1}(0)$ to be split into two tapes: one with tape number 1 and corresponding information c_2, the other with tape number 0 and corresponding information $\neg c_1 \wedge \neg c_2$.

It is easy to verify that at step p_2 the number represented on the $\psi(0)$ tapes is z_2 such that $\|z_2 = (c_1 \vee c_2) \cdot 1 \oplus \neg(c_1 \vee c_2) \cdot 0\| = 1$.

After $p = \max\{p_1, \ldots, p_n\}$ steps, all the stop conditions for $\phi_n(n)$ are reached, thus the tapes of $\psi(0)$ never change, and so we have:

$$\|\tau(n) = \bigvee_{i \leq n} c_i \cdot 1 \oplus \neg \bigvee_{i \leq n} c_i \cdot 0\| = 1$$

This completes the proof. $\diamondsuit$

Acknowledgements.

I thank Professor Giovanni Sambin, who has played a role in connection with this paper even beyond that of an advisor; his interest in my ideas, and his patience to examine several preliminary versions, have been essential to clarify my own understanding, to find suitable definitions and to improve exposition. But of course, if the paper still contains mistakes or obscurities, it is only my responsibility.

I thank also Professor Enrico Pagello, Doctors Nino Trainito, Nicola Guarino, Silvana Badaloni and Paolo Bison that provided valuable discussions.

References.

[B] Bell, J.L., "Boolean valued models and independence proofs in set theory," Second edition, Oxford University Press., 1985.

[H1] Halmos, P.R., "Lectures on Boolean algebras," Van Nostrand, New York, 1963.

[H2] Halmos, P.R., "Measure theory," Van Nostrand, New York, 1950.

[R] Rogers, H., Jr., "The theory of recursive functions and effective computability," McGraw-Hill, 1967.

[S1] Sossai, C., *An extension of Turing machines*, in "The mathematical revolution inspired by computing," J. H. Johnson and M. J. Loomes eds., Oxford University Press, 1990 (to appear).

[S2] Sossai, C., *Learning by abstraction using CET machines*, in preparation.

[TZ] Takeuti, G. and Zaring, W.N., "Introduction to axiomatic set theory," Springer, Berlin, 1971.