Computing as Experimental Thinking

Alberto Pettorossi

DISP, University of Roma Tor Vergata, I-00133 Roma, Italy
alberto.pettorossi@uniroma2.it

1 Introduction

In Physics we construct *models of reality* (also called *theories*) by using mathematical structures and we perform *experiments* to gain confidence on these models. Experiments are also used to improve these models. Indeed, if the measures derived from the experiments do not agree with the expected results, we have to modify our models. Well established theories are, for instance, the theory of thermodynamics, the theory of gravitation, and the theory of relativity.

Theories are used to improve *technical instruments* (such as clocks, thermometers, etc.) and by getting more precise measurements we may improve our theories. Theories are also used to design better *machines* (such as cars, airplanes, etc.) and these machines may determine a higher quality of life. Unfortunately, theories are also used to construct offensive weapons and this may cause undesirable, destructive effects.

A similar interaction between theories, experiments, instruments, and machines, also occurs in Theoretical Computer Science. In this science, which is a relatively young science, we construct *models of computing* and we perform *experiments* to gain confidence on these models and improve them. These experiments may be either *gedanken experiments* or *computer experiments*. As in the case of Physics, we get theories (also called *calculi* or *algebras*). We have, for instance: the type theories, the λ-calculi, the partial recursive function theory, the theory of logic programming, the theory of functional programming, the relational theory of data bases, and the algebras for concurrency.

Theories, calculi, and algebras are used to improve computations, to produce computers with better reasoning properties (with deduction, induction, and synthesis abilities), to allow more powerful representations of knowledge and information. Better computers and better communications may determine a higher quality of life.

The theory of Computer Science and its technological applications are growing very fast and it is somewhat difficult to understand their power and limitations, their capabilities and drawbacks.

Here we would like to examine some issues which are related to the simulation of human intelligence via computers. This study will also give us the opportunity of clarifying a few relevant notions which are used in Computer Science and Logic. In particular, we will briefly present and discuss the following concepts:
(i) *proof* and *deduction* (of a formula in a theory),
(ii) *decidability* (of a theory) and *complexity* (of a proof in a theory),

(iii) *truth* (of a formula in a mathematical structure),
(iv) *expressivity* (of a formal language), and
(v) degrees of *intelligence.*

2 Computations as Constructive Proofs

Computers perform computations by performing proofs in formalized theories. Every computation can be considered as a deduction in a chosen formalized theory which models an observable portion of the reality. We illustrate this idea in the following Table 1, which shows how the multiplication of two natural numbers, say 5 and 3, viewed as an observable portion of the reality, is performed in the usual theory of the binary arithmetics, called *Binary Theory.* One starts off by representing the number 5 by the sequence 101 and the number 3 by the sequence 11. Then, one applies the rules of the Binary Theory, and the sequence 1111 is deduced from the sequences 101 and 11. This deduction corresponds to the desired multiplication because, according to the Binary Theory, 1111 represents the natural number 15, that is, the anti-representation of 1111 is 15.

the observable reality: *Natural Numbers*		proof in the *Binary Theory*
$5\times$	$\rightarrow$ representation $\rightarrow$	$101\times$
$3=$	$\rightarrow$ representation $\rightarrow$	$11=$
		101
		101
result: 15	$\leftarrow$ anti-representation $\leftarrow$	1111

Table 1. A multiplication as a proof in the Binary Theory.
There is a bisimilation between the Natural Numbers and the Binary Theory.

Thus, the multiplication of two natural numbers is a *constructive proof* in the Binary Theory. The deduction of 1111 from 101 and 11 corresponds to the construction of a witness m of the existential quantifier of the formula $\exists m\, 101 \times 11 = m$.

This multiplication example can be generalized. Indeed, any computation over an observable portion of reality, either an external or a mental reality, is a constructive proof, that is, it can be viewed as the construction of a witness of an existentially quantified formula in a formalized theory. This theory should be a suitable theoretical model of the reality of interest. By 'suitable' we mean that between the reality and the theory there exists a *bisimulation* [4]. Basically, the existence of a bisimulation ensures that representations and anti-representations correctly anticipate the changes in the reality through the changes in the formalized theory.

By constructing proofs in formalized theories, computers are capable to simulate the so called *procedural intelligence.* By this notion we refer to the form of intelligence which is the application of deduction rules in a given formal theory. By that form of intelligence computers derive conclusions from premises exactly as people would do. We will argue that this form of intelligence is a low form of intelligence. There are higher forms of intelligence which we will consider in the sequel. Since computers can apply deduction rules in a very fast way, precisely for this ability of being fast, they often have better performance than those of humans and, in this sense, computers may have a higher level of procedural intelligence than people.

3 Synthesis of Programs as Constructive Proofs

Besides procedural intelligence, computers can simulate a different form of intelligence which we call *creative intelligence*. It is, indeed, the case that via constructive proofs in suitable logical theories we can mechanically generate programs which implement some given specifications. This mechanical construction guarantees the correctness of the programs which are produced. But there is a price to pay: the theories in which these constructions are made, are not decidable, in general. Thus, there is no guarantee that the techniques for generating programs are always terminating. This means that in these theories the search for a constructive proof may end up in an infinite computation. Thus, it is often required to have strategies and heuristics to guide the process of proof construction.

Now we present an example of simulation of creative intelligence which can be performed by using computer systems. Suppose that we want to produce a program *Mult* for multiplying two natural numbers, having at our disposal the basic operation of addition, denoted $+$. We are given the specification:

$$\forall u, v \in N \;\; \exists w \in N \;\; \text{such that } w = \underbrace{u + \ldots + u}_{v \text{ times}}$$

The desired program which can be mechanically produced is as follows:

$$w = mult(u,v) \quad \text{where} \quad \begin{array}{ll} 1. & mult(0,v) = 0 \\ 2. & mult(u{+}1,v) = mult(u,v) + v \end{array}$$

This program correctly implements the given specification and, indeed, for instance, we have:

$$mult(1,2) = \{\text{by } 2.\} = mult(0,2){+}2 = \{\text{by } 1.\} = 0{+}2 = \{\text{by } + \text{ definition}\} = 2.$$

We consider that the generation of programs is done by creative intelligence, and this intelligence is 'more advanced' than procedural intelligence, simply because it is usually assumed that to produce programs is more difficult than to perform computations. But the difference between these two forms of intelligence only relies on the properties of the theories where deductions take place. In the case of procedural intelligence deduction is performed within a decidable theory, while in the case of creative intelligence deduction is, in general, performed within an undecidable theory [3]. The discussion of this issue leads us to the following section where we deal with: (i) derivation of formulas, and (ii) decidability of theories.

4 Derivation of Formulas and Decidability of Theories

In formal logic the idea of deriving new facts form old facts is captured by the process of deriving new formulas from old formulas. Some of these formulas are given a particular status, they are called *theorems*. They are formulas which are intended to state true facts.

In order to derive theorems we starts from *axioms* which are the initial theorems, and by applying rules of inference we derive new theorems from old theorems. A theory is a set of theorems. A theory can be presented by giving its axioms and inference rules. In particular, the familiar Propositional Calculus has the following axioms:

1. $A \to (B \to A)$
2. $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$
3. $(\neg B \to \neg A) \to (A \to B)$

and a unique rule of inference, called *Modus Ponens*. This rule of inference has the two premises A and $A \to B$, and one conclusion B, and it is written as follows:

$\frac{A \quad A \to B}{B}$

In the above axioms and inference rule the letters A, B, and C stands for any formula in the theory. The *derivation* of a formula from other formulas in a given formal theory, is done by applying a finitistic rule of inference, i.e., a rule of inference with a finite number of premises. This application takes a finite amount of time and a finite amount of space. In this sense it is a mechanizable, finite computation.

A theory T is said to be *decidable* iff there exists an executable program (or algorithm) P which always terminates such that for any given formula φ, P halts and 'says yes' iff φ is a theorem of the theory. If φ is a theorem of a theory T we write $T \vdash \varphi$.

Some theories may be *semidecidable*. For a semidecidable theory there exists an executable program P such that for any given formula φ, P halts and 'says yes' iff φ is a theorem of the theory. If the execution of P is not terminated yet, one can say nothing about φ being a theorem or not.

There are theories which are not decidable and are not semidecidable.

Since we refer to an executable program P, the notion of decidability of a theory is related to the notion of *computability*. This is a important relationship. It is necessary to agree on a notion of computability, before stating whether or not a theory is decidable. Fortunately, there is a notion of 'intuitive computability' which is sufficiently stable in Computer Science, at least for the set of functions from the natural numbers to the natural numbers. This notion is captured by the Church Thesis which establishes the equivalence of 'intuitive computability' with the formal computability within any of the following calculi: λ-calculus, partial recursive functions, combinatory logic, Turing computations (or Turing machines), type-0 grammars, Markov algorithms, and RAM machines.

We assume, as usually in Computer Science, that Church Thesis holds and under this assumption, we have an unfortunate result which can be informally stated as follows: while the expressivity of a theory increases, its decidability decreases.

Undecidability of theories is related to undecidability of problems. This relationship is based on the fact that any theory, as we have already mentioned, is a set of formulas, and also any problem whose instances have either a yes-answer or a no-answer, can be viewed as a set of formulas, namely those formulas corresponding to the instances of the problem with yes-answer.

Now we present an example of an undecidable problem: it is the Post Correspondence Problem. Let us consider a list of $n(\geq 1)$ pairs of strings of 0's and 1's, say $\langle u_0, v_0\rangle$, ..., $\langle u_{n-1}, v_{n-1}\rangle$. We want to find an integer k and a sequence $\langle i_1, \ldots, i_k\rangle$ of indexes, each of which is an element of $\{0, \ldots n-1\}$, such that $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$. This is an undecidable problem in the sense that it does not exist an executable program P such that: (i) P always terminates, and (ii) for any list of pairs of strings of 0's and 1's, P halts and 'says yes' iff there exist k and a sequence $\langle i_1, \ldots, i_k\rangle$ of indexes, each of which is an element of $\{0, \ldots n-1\}$, such that $u_{i_1} \ldots u_{i_k} = v_{i_1} \ldots v_{i_k}$. The following Table 2 shows an instance of the Post Correspondence Problem.

index	left string	right string
1 :	0011	1
2 :	1	101
3 :	011	1001
4 :	110	1
5 :	1	11

Table 2. An instance of the Post Correspondence Problem.

The sequence of indexes which solves this instance of the Post Correspondence Problem is: $\langle 4, 3, 5, 2, 3, 1\rangle$, and indeed,

	left string						=	right string					
	110	011	1	1	011	0011	=	1	1001	11	101	1001	1
index:	4	3	5	2	3	1		4	3	5	2	3	1

For later reference we recall here that: (i) a formula is said to be *closed* iff every variable occurrence is either universally or existentially quantified, and (ii) a theory T is said to be *complete* iff for every closed formula φ, either $T \vdash \varphi$ or $T \vdash \neg\varphi$ [3].

5 Complexity of Proofs

When performing derivations in decidable theories we have at our disposal a program which always terminates for deciding whether or not a given formula is a theorem, but the computations performed by that program may be very expensive in time or space. Indeed, there is a difference between: (i) what is terminating, i.e., computable in *finite* time, and (ii) what is computable in *reasonable* time.

This notion of being reasonable is called the *complexity* of the derivations (or proofs) in the theory under consideration.

Now we give two examples of complexity of derivations in decidable theories.

The first example is taken from the Presburger Arithmetics $(Z, +, =, 0, 1)$, which is the theory of the set Z of integer numbers with the addition. A formula of that theory is, for instance, $\forall x\, \exists y\, \exists z\, (x{+}z = y \wedge \exists w\; w{+}w = y)$, which expresses that for any integer x there exists a larger even integer y. In this theory there are formulas of length n whose proofs require $O(2^{2^{2^{cn}}})$ steps for some constant $c\,(>0)$. Thus, the time complexity is exponentially high, and even if we have at our disposal very powerful computers, their ability of deciding formulas of Presburger Arithmetics may be too weak.

The second example is taken from the Weak Monadic Second Order Theory with 1 Successor, usually denoted by WS1S. In this theory, for any $k\,(\geq 0)$ there exists a formula of length n whose proof requires $O(2^{\cdot^{\cdot^{\cdot^{2^{cn}}}}})$ steps for some constant $c\,(>0)$, and the height of the tower of 2's is k.

These very high complexity measures tell us that the ability of simulating creative intelligence as well as procedural intelligence, via computations within formal theories is very much limited in practice.

6 Truth in a Mathematical Structure

Here we discuss a few basic issues about the notion of truth in mathematical structures. We begin by asking ourselves the following question: Is 1+1 equal to 2 (as in ordinary arithmetics) or equal to 10 (as in binary arithmetics)?

To distinguish between these two answers, and in general, to establish what is a true statement in a mathematical structure, say M, we may characterize that structure by a set of axioms and inference rules, collectively called $AR(M)$. Then we say that φ is true in a structure M iff φ can be deduced using the axioms and the inference rules in $AR(M)$, that is, $AR(M) \vdash \varphi$.

In order to illustrate these ideas let us now consider the following example which refers to boolean algebras.

A boolean algebra is a non-empty set B together with two binary operations $\vee$ and $\wedge$, one unary operation $\neg$, and two nullary operations 0 and 1. A boolean algebra is denoted by the 6-tuple $\langle B, \vee, \wedge, \neg, 0, 1\rangle$. Here is the set *Bool* of axioms for boolean algebras (in these axioms it is assumed that all free variables x, y, and z are implicitly universally quantified at the front):

1.a	$x \wedge y = y \wedge x$	1.b	$x \vee y = y \vee x$
2.a	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	2.b	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
3.a	$x \wedge (\neg x) = 0$	3.b	$x \vee (\neg x) = 1$
4.a	$x \vee 0 = x$	4.b	$x \wedge 1 = x$
5.	$0 \neq 1$		

We say that a property φ is true in any boolean algebra iff φ can be deduced from the set *Bool* of axioms by using the replacement of *equals by equals* as the only rule of inference. This rule allows us to deduce $E[a] = E[b]$ from $a = b$, for any given expression E.

Here is the proof that in every boolean algebra it is true that $\forall x \in B\ 1 = x \vee 1$. For brevity we omit to write '$\forall x \in B$' and we write within curly brackets the axioms used for the deduction.

$$\begin{aligned} 1 = \{3.\mathrm{b}\} = x \vee \neg x = \{4.\mathrm{b}\} = x \vee (\neg x \wedge 1) = \{2.\mathrm{b}\} = (x \vee \neg x) \wedge (x \vee 1) = \\ = \{3.\mathrm{b}\} = 1 \wedge (x \vee 1) = \{1.\mathrm{a}\} = (x \vee 1) \wedge 1 = \{4.\mathrm{b}\} = x \vee 1. \end{aligned}$$

Notice, however, that it is not always the case that given a structure M, we can indeed construct $AR(M)$ so that every statement φ which is true in M can be derived from $AR(M)$. Here we cannot provide more evidence for this fact (see the Gödel-Rosser Incompleteness Theorem [3]). Let us simply recall that the mathematical structures are, indeed, very rich structures and not all their properties can be captured by deductive systems based on axioms and inference rules. Obviously, to formalize this claim, we should give a formal definition of what we mean by axioms and inference rules. But we do not do that here.

In order to formalize the notion of truth in a mathematical structure we may follow the approach of defining in an independent way: (1) the notion of what we mean for a statement φ *to be true* in a structure M, denoted $M \models \varphi$, and (2) the notion of what we mean for a statement φ *to be derivable* from the axioms and the rules of inference of $AR(M)$, denoted $M \vdash \varphi$. Then, we may study the relationship between these two notions. It turns out that for a suitable choice of $\models$ and $\vdash$, the two notions coincide for the First Order Predicate Calculus (see Gödel Completeness Theorem [3]).

Now let us see Tarski's approach to the definition of truth of a formula φ in a mathematical structure M. Tarski gives rules for establishing, for instance, that $N \models \exists x\, 6/2 = x$ holds and $Even \models \exists x\, 6/2 = x$ does not hold, where N denotes the natural numbers and *Even* denotes the even natural numbers. We cannot give the details of Tarski's approach (the reader may refer to [3]). Given a formula φ, its truth depends on: (i) the choice of a domain (for instance, the natural numbers N), (ii) the interpretation of the symbols in the formula (for instance, $/$ is the integer division from $N \times N$ to N, and $=$ is the equality relation, subset on $N \times N$), (iii) the binding of its free variables, if any, to elements of the domain, and (iv) the 'lifting to metalevel' of the logical operators and the quantifiers (for instance, for the operator $\wedge$, we establish that: $N \models \varphi_1 \wedge \varphi_2$ iff $N \models \varphi_1$ and $N \models \varphi_2$). This last Point (iv) looks like playing a trick because, for instance, one uses the metalevel, linguistic connective 'and' for providing the exact meaning of the object level, formal operator $\wedge$. Likewise, for the other operators and the quantifiers.

Despite some inherent weaknesses, the Tarskian approach to the formal definition of the notion of truth is quite stable. From the notion of a formula φ being true in a mathematical structure (or interpretation) I, denoted $I \models \varphi$, one derives the notion of *validity* of a formula φ, denoted $\models \varphi$. We say that a formula φ is *valid* iff for all interpretation I, $I \models \varphi$. For instance, $((x = 2) \Leftrightarrow (x = 2 \wedge x = 2))$ is a valid formula.

Tarskian approach to the notion of truth works for many different logics, either modal or temporal ones.

In modal logics we have an operator corresponding to *must*, denoted $\Box$, and an operator corresponding to *may*, denoted $\Diamond$. There is a duality between them. For instance, $\Diamond\neg\varphi$ is equivalent to $\neg\Box\varphi$.

In temporal logics there are various operators for expressing time relationships. For instance, we can express *now*, *in-the-future*, *in-the-past*, etc. We can also refer to a *linear* time logic, where events take place in a linearly ordered structure, or a *branching* time logic, where events take place in a tree-like structure. The flow of time may be either discrete or dense or continuous.

7 Expressivity versus Decidability

In this section we present a sequence of well-known logic calculi which have an increasing descriptive power, also called *expressivity*. More expressive calculi allow us more elaborated forms of reasoning or deductions. In relation to this sequence we want to indicate that the increase of expressivity is done at the cost of decreasing the level of decidability. This is a very important feature of formal calculi and it entails that if we want to express or reason about more detailed properties then we get a reduced ability of establishing the truth of these properties.

The first calculus in the sequence is the *Propositional Calculus* [3]. Its set of sentences consists of atomic propositions (as many as the natural numbers) which are put together using *and* ($\wedge$), *or* ($\vee$), and *negation* ($\neg$). Thus, for instance, we may write: $p_0 \vee p_1$ and we mean that either p_0 or p_1 holds.

The second calculus in the sequence is the so called *First Order Predicate Calculus* [3]. It extends the Propositional Calculus by allowing variables and quantification over variables. For instance, we can write: $\exists x\, human(x) \wedge x = Socrates$ and we mean that there is an individual which is *human* and it equal to *Socrates* (or, equivalently, the individual *Socrates* is *human*.) We do not enter here into the discussion concerning the epistemological status of the existential quantifier and, in particular, whether or not the existential quantifier asserts the existence of an individual such as *Socrates* in our example above.

The third calculus in the sequence is the so called *Peano Arithmetics* (*PA*, for short) [3] which is a particular First Order Predicate Calculus with: (i) an extra binary predicate symbol $=$, (ii) one extra constant 0, (iii) three extra function symbols: s (unary), called *successor*, $+$ (binary), called *plus*, and $\times$(binary), called *times*, and (iv) the following extra axioms:

E1. $\forall x\; x = x$
E2. $\forall x \forall y\; (x = y \rightarrow (A(x,x) \rightarrow A(x,y)))$
PA1. $\forall xyz\; (x = y \rightarrow (x = z \rightarrow y = z))$
PA2. $\forall xy\; (x = y \rightarrow (s(x) = s(y)))$
PA3. $\forall x\; 0 \neq s(x)$
PA4. $\forall xy\; (s(x) = s(y) \rightarrow x = y)$
PA5. $\forall x\; x + 0 = x$
PA6. $\forall xy\; x + s(y) = s(x + y)$

PA7. $\forall x\ x \times 0 = 0$
PA8. $\forall xy\ x \times s(y) = (x \times y) + x$
PA9. for any formula $A(x)$, $(A(0) \wedge \forall x\,(A(x) \rightarrow A(s(x)))) \rightarrow \forall x\, A(x)$

In the axiom E2 the formula $A(x,y)$ stands the formula $A(x,x)$ where some free occurrences of the variable x have been replaced by the variable y and y is free for x in $A(x,x)$. From E1 and E2 we get the usual properties of equality: (i) reflexivity, (ii) symmetry, (iii) transitivity, and (iv) substitutivity, that is, $\forall x \forall y\ (x = y \rightarrow (A(x,x) \leftrightarrow A(x,y)))$. In particular, from E2 it follows that $\forall x \forall y\ (x = y \rightarrow (A(x,y) \rightarrow A(x,x)))$.

The fourth and last calculus in the sequence is the so called *Second Order Predicate Calculus* [2]. It extends the *First Order Predicate Calculus* by allowing quantifications over predicate symbols and individual variables as well. For instance, the following is a formula of the Second Order Predicate Calculus: $\forall A\,(A(0) \wedge \forall x\,(A(x) \rightarrow A(s(x)))) \rightarrow \forall x\, A(x)$, where A is variable ranging over predicate symbols and x is a variable ranging over individuals. We cannot go into more details here and we refer to [2].

The following Table 3 illustrates that the increase of expressivity determines the decrease of decidability and the loss of the equivalence between $\vdash$ and $\models$.

---

1. *Propositional Calculus* (low expressivity).
$\vdash$ is the same as $\models$, that is, for every φ in the language of Propositional Calculus, $\vdash \varphi$ iff $\models \varphi$.
It is decidable: in finite time, via a Turing computation, we decide wheter or not a formula is a theorem.

2. *First Order Predicate Calculus.*
$\vdash$ is the same as $\models$, that is, for every φ in the language of First Order Predicate Calculus, $\vdash \varphi$ iff $\models \varphi$.
It is semidecidable: there exists a Turing machine M such that for all formulas φ, φ is a theorem iff M halts and says 'yes'.

3. *Peano Arithmetics.*
It is incomplete: there exists a sentence φ such that neither φ nor $\neg\varphi$ can be derived.
$\vdash$ is the same as $\models$, that is, for every φ in the language of Peano Arithmetics, $PA \vdash \varphi$ iff $(\forall I\,(\text{if } I \models PA \text{ then } I \models \varphi))$.
It is semidecidable.

4. *Second Order Predicate Calculus* (high expressivity).
It is incomplete.
$\vdash$ is different from $\models$.
It is not decidable and it is not semidecidable.

Table 3. Logic Calculi of increasing expressivity:
1. Propositional Calculus, 2. First Order Predicate Calculus,
3. Peano Arithmetics, 4. Second Order Predicate Calculus.

This table shows a negative result: when using languages with higher expressivity or when looking for a higher quality of reasoning (i.e., going from Propositional Calculus to Second Order Predicate Calculus), at a certain point we are forced to give up the ability of deducing via computations everything which is true about the structure we want to describe (that is, we loose the equivalence between $\vdash$ and $\models$). We also have to give up the ability to decide whether a given property is true (that is, we loose decidability).

8 Various Degrees of Intelligence

In Sections 2 and 3 we have already illustrated two degrees of intelligence: (i) the *procedural intelligence*, which is basically the process of applying a set of rules in a decidable theory according to a particular executable program (in the same way as a Turing machine would do), and (ii) the *creative intelligence* which is analogous to the procedural intelligence, but the executable program works in an undecidable theory. In order to produce significant results, creative intelligence requires strategies. Obviously, in the case of creative intelligence, there is no guarantee that significant results can be produced in *all* cases. In other words, there is no complete strategy.

These two degrees of intelligence belong to a first level of intelligence, where computations take place within formal theories. These degrees of intelligence are both mechanizable by using a Turing machine. The usefulness of this mechanization depends on the efficiency of the programs and strategies at our disposal. Here, again, we face a negative result because there are theoretical limitations to the maximum efficiency one can get. For instance, within Propositional Calculus, we cannot decide whether or not a formula is a theorem in less than exponential time, in general.

A different form of intelligence from the procedural intelligence and creative intelligence, is the *intelligence which generates new theories from old theories.* It is not confined within a given theory and, indeed, it allows us to generate new theories. This intelligence is not mechanizable, in general. It uses some form of induction (not simply deductions within theories) and it can be partially simulated via Artificial Intelligence techniques. Notice, however, that if we have produced a new first order theory for describing a given reality, the problem of deciding whether or not this first order theory is consistent (that is, *false* is not deducible in the theory) is not mechanizable.

There is also another form of intelligence, namely, the *intelligence which discovers the adequacy of a theory to a given reality.* This form of intelligence is not mechanizable, at least when the observable phenomena of the reality are sufficiently detailed. In other words, if through our theory we want to describe a class of sufficiently detailed observations then it does not exist an executable program which ensures that our theory is adequate for all those observations.

A final form of intelligence is related to the understanding of the *inner meaning of things* and, in a sense, this form of intelligence goes beyond the things themselves. This intelligence views the external reality in relation with people and it reflects on the values and the goals of the human actions. This intelligence is the substratum for free-will and self-conscience, and these two manifestations of human nature are not mechanizable, simply because there is no mathematical model for them. For this reason I hold that this degree of intelligence can never be fully simulated by machines or communicated to machines.

Before closing, let us briefly present the approach described in [1] where the so-called *observe-think-act cycle* is proposed as a technique for machine simulation of intelligent behaviour. Through sensors the machine (or robot) gets stimuli from the environment during the *observe* phase, then during the *think* phase, the machine enters a computational activity where it computes the action to take for achieving a given desired goal starting from the present state of the environment, as it has been sensed in the *observe* phase. This computational phase is usually realized via deductions in suitable logical theories. This phase allows both a *reactive thinking* and a *proactive thinking*. Indeed, reactive thinking is possible because the machine may compute actions which are triggered by the values of the present situation of the environment, and proactive thinking is also possible by taking into account the desired goals to be achieved. In the final phase, i.e., the *act* phase, the computed actions are performed via actuators and these actions hopefully modify the external environment as desired. Obviously, this model of intelligent behaviour is subject to the limitations of all computations within theories which we have indicated in the previous sections, and in particular, it is subject to the decidability limitations and the complexity limitations. Thus, the simulation of intelligence by using the observe-think-act cycle can never be completely achieved in practice.

9 Acknowledgements

Many thanks to Prof. Gennaro Cicchese, Prof. Piero Coda, and Prof. Sergio Rondinara for their invitation to the Sefir Congress: 'Le implicazioni ontologiche dei saperi scientifici', held in Roma (Italy) at the Pontifical Lateran University in January 2003. Their comments and suggestions, as well as those of the other participants, have been very helpful and stimulating. Dr. Maurizio Proietti of the IASI Institute of the Italian National Research Council in Rome helped me to clarify a few crucial points.

References

1. R. A. Kowalski. Artificial intelligence and the natural world. *Cognitive Processing*, 2, 2001.
2. Z. Manna. *Mathematical Theory of Computation*. MacGraw-Hill, 1974.
3. E. Mendelson. *Introduction to Mathematical Logic*. Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, California, Usa, Monterey, California, Usa, 1987. Third Edition.
4. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.