THE CYBERNETIC FOUNDATION OF MATHEMATICS
I.THE CONCEPT OF TRUTH

Valentin F. Turchin

The City College of New York.
Computer Science Department

This paper is the first part of a compressed exposition of a new approach to the foundation of mathematics, which is referred below as *The Cybernetic Foundation*. A complete exposition will be found in a book which is being prepared for publication.

1. Introduction

At the present time, set theory serves as the basis for mathematical constructions and proofs. The intuitive interpretation of set theory is, as it has always been since Georg Cantor, who invented it, that sets have, somehow, an objective existence different form the way material objects exist. Moreover, sets may be "actually infinite". This Platonist concept goes against the letter and the spirit of modern philosophy widely shared by people of science.

We cannot rely on our intuition when dealing with "actually infinite" sets; this leads to inconsistences ("paradoxes"). Cantor's concept_of set, which includes the idea that "some infinite sets are more infinite than others", is counterintuitive. In itself, the situation when a theoretical concept defies our intuition is not unacceptable. To take a famous example from calculus, there are functions which in every point are continuous but not differentiable. However, such a function is a concept constructed logically from more primitive conceptual units, and we can satisfy our intuition that the construction is flawless. When the most primitive and fundamental conceptual units are counterintuitive, so that you can neither derive them, nor find ism based on the concept of actual infinity is not only incomprehensible, but simply wrong. It is conceivable that if this really is the case, the mathematicians could have developed their set-theoretical intuition in response to the real, and not the proclaimed, objects of set theory.

Now look at the set-theoretical foundation of mathematics from the angle of consistensy. Working with set theory, one gets an intuitive impression, maybe even a certainty, that it is non-contradictory, consistent. But its consistency has never been proved. This is very strange, if we come to think about it. Axiomatic set theory in the Zermelo-Fraenkel form rests on ten axioms, most of which are far from being elementary, or primitive. Taken all together, they make up a still less primitive whole. It is inconceivable that our intuition can perceive the consistency of this whole without basing itself on some simple, primitive, intuitively consistent concepts and truths. We come to believe, therefore, that such primitive and intuitively unquestionable truths must exist. To separate them and to express in terms of them the ZF axioms, would be to prove the consistency of set theory. From Goedel's theorem we know, however, that it is impossible to prove the consistency of set theory by means which can be formalized in set theory. Hence the primitive concepts and truths underlying set theory must be very unusual, strange, because they must be non-expressible in set theory, while we habitually entertain the idea that in set theory we can express everything that can be subject to rigorous mathematical treatment. A theory based on these concepts must be equally 'strange'. To use an expression popular among physicists and coined by Niels Bohr, such a theory must be 'crazy enough'.

The Cybernetic Foundation is such a 'crazy' theory. Its basic concepts, although puzzling at the beginning, upon some thought become self-evident, and the legitimacy of their use in proofs -- intuitively doubtless. At least, this is the author's view. This theory leads to a full acceptance of the formalism of set theory, but interprets it in agreement with the principles of constructivism, using only the idea of potential, but not actual, infinity. This becomes possible because of two new ideas which form the basis of our theory.

The first idea is to base the semantics of the mathematical language on the cybernetical concept of knowledge. According to this concept, to say that a cybernetic system (a human being, in particular) has some knowledge is to say that it has some models of reality. In Cybernetic Foundation we consider mathematics as the art of constructing linguistic models of reality. An analysis of the concept of model shows that a model is, essentially, a generator of predictions. We formalize a prediction as the statement that a given process is finite. We declare a proposition meaningful if, and only if, it can be interpreted as a generator of predictions.

The second major idea behind the Cybernetic Foundation is the introduction of the subject of knowledge in mathematics, which leads to a new kind of processes. We call them *metamechanical*. A metamechanical process is initiated and maintained by a mechanical device like a Turing machine in interaction with the subject of knowledge, i.e. the user of the device. The class of metamechanical processes is wider than the class of processes which can be generated by a Turing machine, or by any other autonomous mechanical device.

Our main result in this paper (Part II) is the proof of the consistency of the full set theory, as formalized by the Zermelo-Fraenkel axioms. The philosophy of science which underlies the Cybernetic Foundation is laid out in [Turchin 1977].

2. The Refal Machine

An important step towards putting mathematics on the empiricist track was made by Alan M. Turing. In 1936, he introduced in mathematics an abstract device, or rather a class of devices, which became known as *Turing machines*. The idea was to make mathematical computation, in the widest sense, an object of mathematical study.

Our formalism is based on the concept of the *Refal machine*.

'Refal' is the acronym for *REcursive Functions Algorithmic Language*, a computer programming language which was developed by the author and co-workers in 1966-1970 and is implemented on several computer systems, including the IBM/370. Refal as an algorithmic language is conceived to be simple enough to allow mathematical treatment but still successful as a practical programming language for such fields as artificial intelligence and word processing. A guide for programming in Refal can be found in [Turchin, 1980].

The reason why we chose Refal, and not the usual in this context Turing machine, is that our goal is a complete formalization of mathematics, so that definitions of mathematical concepts could actually be used as programs and run in the computer. Our Refal formalism allows a clear and concise definition of processes and machines, and hierarchical construction of machines which control machines; this would be very cumbersome if we used Turing machines only. In the present paper we limit ourselves to a rather informal definition of Refal, and use a semi-formal notation usual in mathematics. In the full text of the book, this notation is given a formal interpretation, an all machines defined here in words are formally defined in Refal.

We shall discuss three aspects of Refal, in the order of increasing generality.

Refal can be seen as a language of semantic descriptions. The following line:

(1) <ACM> → ASSOCIATION FOR COMPUTING MACHINERY

is a *sentence* of Refal. The angular brackets are *concretization brackets*. They enclose a linguistic object which must be concretized, that is replaced by linguistic objects which in some sense are closer to the ultimate reality. The sentence (1) expresses the expansion of the acronym ACM. It consists of the left side and the right side separated by the arrow →. The Refal machine, i.e. the device that "understands" Refal, takes this sentence as the instruction to replace <ACM> by 'ASSOCIATION FOR COMPUTING

MACHINERY'. The letters A,C,M,A,S,S,O, ... etc. will be referred to as *symbols*. Angular brackets are *special signs* of Refal, not symbols.

Consider another sentence:

(2) <THE FIRST SYMBOL OF s_1e_2> → s_1

It defines what the first symbol of an expression is, and can be translated as: the first symbol of an expression which consists of a symbol s_1 after which an expression e_2 immediately follows, is s_1. Here s_1 and e_2 are free variables. The fromer is a *symbol variable* (s-variable, for short), the latter an *expression variable* (e-variable). This sentence is used in the following way. Suppose we observe the expression

(3) <THE FIRST SYMBOL OF APPLE>

To see whether the sentence (2) is applicable, compare its left side with (3). If it is possible to give such values to the free variables that the left side of (2) becomes identical to (3), then the sentence (2) is applicable to the concretization of (3). In assigning values to variables we must remember that an s-variable must take as its value exactly one symbol, while an e-variable can take any expression. Clearly, (2) is applicable if s_1 takes the value A, and e_2 the value PPLE. To apply a sentence means to replace the expression (3) by the right side in which the values of the free variables are substituted. The result is A. We have performed one step of the Refal machine.

Now let us look at Refal from another angle, namely as the language of recursive functions. Let us change the string 'THE FIRST SYMBOL OF' for the single symbol F:

(4) <F s_1e_2> → s_1

We can see the replacemnt of <F APPLE> by A as the evaluation of a function call. Then (4) defines the function F whose value is the first symbol of its argument. The angular brackets should be called *evaluation brackets*. They enclose the expressions which must be understood as function calls and evaluated. <F *A*> corresponds to F(*A*) in the usual notation.

Besides angular brackets, which indicate evaluation, we use in Refal usual round brackets (parentheses): They serve a different purpose: to give a structure to expressions. Any sequence of symbols and parentheses in which the parentheses are properly paired is a legitimate expression in Refal. Parentheses, like concretization/evaluation brackets, are not symbols, but special signs. Here are examples of Refal expressions (separated by commas):

A, ABC, A+B(), (BBB+(**))(())+-

An empty expression (just nothing) is also a legitimate expression. The argument of a Refal function can always be considered as one expression. If we want to define a function of several arguments, we use parentheses to combine them into one expression, so that it could be uniquely broken down into the original constituents, when necessary. For instance, the function which concatenates its two arguments can be defined as

$$<\underline{conc}\ (e_x)(e_y)> \;\rightarrow\; e_x e_y$$

Here <u>conc</u> is the name of the function. Syntactically, <u>conc</u> is one symbol; composite symbols, like this one, are formed by underlining a group of letters and digits.

The Refal machine has two information storages: the *program field* and the *view-field*. The former contains a list of sentences (program), which is loaded into the machine before the run and does not change during the run; the latter contains an expression which changes in time as the machine works, thus giving rise to a process.

Consider the following group of sentences:

(5.1) $\langle$<u>chpm</u> + $e_x\rangle \rightarrow$ - $\langle$<u>chpm</u> $e_x\rangle$
(5.2) $\langle$<u>chpm</u> $s_a e_x\rangle \rightarrow s_a \langle$<u>chpm</u> $e_x\rangle$
(5.3) $\langle$<u>chpm</u> $(e_a)e_x\rangle \rightarrow (e_a)\langle$ <u>chpm</u> $e_x\rangle$
(5.4) $\langle$<u>chpm</u>$\rangle \rightarrow$

It defines the function <u>chpm</u>, 'change plus to minus'. If this function is applied to an expression, its value will be the result of the replacement of every sign '+' on the top level of the bracket structure in the argument by the sign '-'. Add these sentences to the program field of the Refal machine. Put in the view-field the expression <<u>chpm</u> C+(A+BX)>, and turn the machine on.

The Refal machine will evaluate this function call by steps, each step being an application of one sentence. It will try to apply sentences *in the order they are listed.* When a sentence is found applicable, it is applied, and this is the end of the step; on the next evaluation step, the Refal machine will try to apply sentences starting with the first one again.

Making the first step, the Refal machine tries to apply the sentence (5.1), but of course fails, because the argumant does not start with '+'. Then it tries to apply (5.2) and this time succeeds. The view-field becomes: C<<u>chpm</u> +(A+BX)> . Function <u>chpm</u> calls itself recursively. The results of the further steps are as follows:

C-<<u>chpm</u> (A+BX)>	by (5.1)
C-(A+BX)<<u>chpm</u>>	by (5.3)
C-(A+BX)	by (5.4)

The last expression has no evaluation brackets. The Refal machine stops. The content of the view-field is the result of the evaluation.

The third, and the most general, view of the language Refal and the Refal machine is as a framework for *the linguistic representation of the world*. We see the world as the interplay of various *processes*, which involve various *objects*. We can change objects ourselves, thereby giving rise to processes. We also can create and start *machines*, which maintain processes autonomously. We use the Refal machine to define linguistic processes which serve as linguistic models of natural phenomena. The concepts of object, process, and machine will be considered primary and given to us intuitively. We can only define them informally for clarification, and characterize their relationship.

A process is thought of as a time sequence of objects, while an object is a time section, or a momentary picture, or a *stage*, of a process. A machine is something that gives rise to a process when given an object or a number of objects (the *input*). An object can also be seen as a special case of process: such that all its stages are the same.

This gives us one more name for the angular brackets in Refal: *activation brackets*. They distinguish a process from an object. An expression enclosed in activation brackets, e.g. <ABC>, represents the current stage of a process, and will be referred to simply as a *process*. Later in time <ABC> may turn (be turned by the Refal machine) into something else, say <ABCD>, as the process develops. An expression which does not include activation brackets will be referred to as *passive*; it represents an *object* that does not change in time. Change comes only from activation brackets.

Suppose we want to define the process of the growth of a string of characters A , i.e. a process whose first stage is empty, then A , then AA , then AAA , etc. How can we do that using the Refal machine?

We know that the representaion of a process in the Refal machine must be enclosed in activation brackets. One possibility is to represent the consecutive stages of our process simply by <> , <A> , <AA> , etc. But it is a better practice to put a tag (a name) on every process, so as to be able to have definitions of different processes without unintended interference between them. Any object expression may serve as a tag, and in the simplest case it will be one symbol. Let us choose the same format as in function representation, i.e. agree that the tag will always be placed at the left end of the process expression, immediately after the opening activation bracket. Let symbol α be the tag for our process. Then <α> will be the initial stage, <αA> will be the next stage, etc. One sentence:

$$\langle\alpha\ e_x\rangle \rightarrow \langle\alpha\ e_x A\rangle$$

in the program field of the Refal machine will define the process. To initiate it, we put <α> in the view-field and start the machine. After the first step the view-field will be <αA> , then <αAA> , then <αAAA> , and so on infinitely. Computation of function chpm is an example of a finite process.

Because different programs can be loaded into the program field, we can use the Refal machine as a *metamachine* through which to define various specific machines. Our concept of a linguistic machine is related but not identical to the concept of a recursive function on the set of object expressions. A recursive function is considered undefined if the process of computation for a given argument is infinite; and if the process is finite then it is only its result that matters, not the process. When we are speaking of a machine in this work, it is exactly *the process* we are interested in, and it may be either finite or infinite.

A machine is defined by specifying: (1) a general Refal expression *F* called *the format* of the machine, and (2) a Refal program, which is its *definition*. Substituting some values for the variables in *F*, we receive a process expression which is then put into the view-field of the Refal machine which is loaded with program *P*.

Let us consider less trivial examples than those above. In the unary number system, where zero is represented by Ø, one by Ø1, two by Ø11, etc., the adding machine with the format $\langle +(e_x)e_y\rangle$ can be defined by the program

$$\langle +(e_x)\emptyset\rangle \rightarrow e_x$$
$$\langle +(e_x)e_y 1\rangle \rightarrow \langle +(e_x)e_y\rangle 1$$

With the input values Ø1 for e_x and Ø11 for e_y this machine will generate a finite computation process which ends with Ø111. We could define an equivalent machine choosing a different format, e.g., $\langle +(e_x)(e_y)\rangle$, or $\langle \underline{adde}_x, e_y\rangle$, etc.

As an example of the use of nested activation brackets, we define an adding machine for binary numbers:

$$\langle \underline{add}(e_x Ø)e_y s_1\rangle \rightarrow \langle \underline{add}(e_x)e_y\rangle s_1$$
$$\langle \underline{add}(e_x 1)e_y Ø\rangle \rightarrow \langle \underline{add}(e_x)e_y\rangle 1$$
$$\langle \underline{add}(e_x 1)e_y 1\rangle \rightarrow \langle \underline{add}(\langle \underline{add}(e_x)1\rangle)e_y\rangle Ø$$
$$\langle \underline{add}(e_x)\rangle \rightarrow e_x$$
$$\langle \underline{add}()e_y\rangle \rightarrow e_y$$

The format is $\langle \underline{add}(e_1)e_2\rangle$. (Note that the variables we choose to represent formats are not related in any way to the variables used in programs; neither are variables in different sentences of the program. But we usually keep to the same variables as a matter of convenience). When there are more than one pair of activation brackets in the view-field, the activation proceeds from left to right and from within out. The active subexpression which is picked up for evaluation at each step is the leftmost of those which have no activation brackets inside.

3. Searches and Generators

We shall deal with processes of two kinds: searches, and generators.

A *search* is a process each stage of which is either of the form $\langle E\rangle$, where E is an expression, or passive. The latter case takes place, obviously, at the end of a finite search. The terminal stage of a search will be referred to as its *result*. An infinite search produces no result. A search, as the name suggests, is a process which you would typically initiate in order to find (construct) a certain object: the result of the search.

A *list* is an expression of the form

$$(E_1)(E_2)\ \ldots\ (E_n)$$

where *n* can be any number including zero (an empty list). A generator is a process each stage of which is either *L*<*E*> or *L*, where *L* is a list of object expressions. The subexpressions E_1, E_2 ... etc. which appear in the view-field at any stage of a process-generator *G* are said to be generated by *G*. A trivial example of a generator is simply a list of object expressions, e.g. (A)(B)(C), which generates symbols A, B, and C, and stops the Refal machine before it has a chance to make a single step. We create generators in order to generate *sets*. A finite set can be represented by an object: the list of its members. An infinite set can be defined only through an actual process. For example, we can construct a generator of all natural numbers represented in the unary form, as above, by defining the num machine as follows:

$$\langle \underline{\text{num}}\ e_x \rangle \rightarrow (e_x)\langle \underline{\text{num}}\ e_x 1\rangle$$

The process <num Ø> is a generator of all natural numbers. The process <num *N*> generates the set of all numbers starting from *N*.

The process <+(Ø1)Ø11> is neither a search, nor a generator. Machines like + , which gradually build up the result in the view-field, are very convenient when programming in Refal, but in the part of our theory that interprets logic and axiomatic mathematics it is easier to manipulate processes if we restrict ourselves to searches and generators. This does not lead to any loss of expressive power of the language. Every machine which is constructed to compute something can be slightly modified so that it initiates a search for the desired result. To achieve that, it is sufficient to replace in the program every right side *R* which is neither <*E*> nor passive, by <out *R*>, where the function out is defined by:

$$\langle \underline{\text{out}}\ e_x \rangle \rightarrow e_x$$

Thus the definition of the addition of unary numbers will become:

```
<+(e₁)Ø>     →   e₁
<+(e₁)e₂1>   →   <out<+(e₁)e₂>1>
```

Now the process of computing Ø1+Ø11 is a search:

```
<+(Ø1)Ø11>
<out<+(Ø1)Ø1>1>
<out<out<+(Ø1)Ø>1>1>
<out<out Ø11>1>
<out Ø111>
Ø111
```

Parallel execution of processes plays an important role in engineering and in our mental pictures of the world. It takes a prominent place in our theory. We can simulate parallel execution of processes in our sequential Refal machine, but definitions in Refal will be much more readable if we have the simulation "on the hardware level" so to say, i.e. if we somewhat expand the abilities of the Refal machine. Therefore, in addition to the familiar form of a Refal sentence:

$$L \to R$$

we allow the following two sentential forms:

$$(s) \qquad L \to s\left|\begin{array}{l} R^1 \\ \\ R^2 \end{array}\right.$$

and

$$(g) \qquad L \to g\left|\begin{array}{l} R^1 \\ \\ R^2 \end{array}\right.$$

When a sentence of the form (s) is applied, the Refal machine creates two auxilliary view-fields. It puts R^1 into one of them, and R^2 into the other (after the substitution of values for variables, as usual). Then it runs processes R^1 and R^2 in parallel. The moment any of them comes to an end, the Refal machine takes its result, substitutes it for the expression under concretization (recognized as L) in the original view-field, and resumes the running of the process in it.

When a sentence of the form (g) is applied, the Refal machine, again, creates two auxiliary view-fields and runs them simultaneously. The interaction between branches, however, is organized differently in this case. Each time that any of the branches produces a list of members, this list is extracted from the branch and placed at the left edge of the projection of L in the main process. The execution of the branch processes goes on as far as at least one of the branches is active. The effect is that every member produced by R^1 or R^2 will be produced by the generator which used sentence (g).

In the Refal machine, symbols and structure brackets (parentheses) serve to create *object expressions*, which represent objects of the external world. Variables and activation brackets can be seen as functional details of the machine itself, which help to perform operations on objects. Therefore, if we are (and we are) to define in Refal processes and machines dealing with parts of the Refal machine, namely the contents of the memory field and the view-field, we need a representation of these parts in the form of object expressions. Such a representation will be called a *metacode*. The metacode we are going to use is defined in the following table:

In the Refal machine	In the metacode
s_I	*SI
e_I	*EI
<	*(
>	)
*	*V
S	S

Here S stands for any object symbol distinct from the asterisk * ; it is represented in the metacode by itself. The asterisk '*' is singled out for representation of special signs in the metacode. One can see that our metacode transformation has a unique inverse transformation. Speaking about linguistic objects and their metacode representations we shall denote by $\uparrow X$ the metacode of X. The inverse transformation will be denoted by $\downarrow$, so that $\uparrow\downarrow X$ is X. The range of the signs $\uparrow$ and $\downarrow$ is the Refal term that follows. Thus, $\uparrow(e_1+e_2)$ is (*E1+*E2), while $\uparrow e_1+e_2$ is $*E1+e_2$.

A program consisting of sentences $Z_1,Z_2, \ldots ,Z_n$ will become

$$(\uparrow Z_1)(\uparrow Z_2) \ \ldots \ (\uparrow Z_n)$$

in the metacode. To give an example of metacode transformation, the program for the '+' machine above will be transformed into

```
(*(+(*EX)0) → *EX)  (*(+(*EX)*EY1) → *(+(*EX)*EY)1)
```

We need a metacode in order to construct machines controlling machines. Suppose, for example, that we want to run two machines, M_1 and M_2, in parallel, by alternating the steps in M_1 and M_2. A machine which can do it must be able to simulate one step of the Refal machine running the processes initiated by M_1 and M_2. We denote this function <step e_x>; its definition, of course, depends on the program stored in the program field. The argument e_x of this function cannot be M_1 or M_2, because these expressions, being active themselves, escape the control of step. Let M_1 be <F(ABC)>. If we form: <step <F(ABC)>> , then the process <F(ABC)> will be run first until it becomes passive (if ever), and then function step will be applied to the result. When we want to control a process, the expression representing it must be metacoded before substitution:

```
<step ↑<F(ABC)>>  =  <step *(F(ABC))>
```

Function step is defined in such a way that <step ↑*P*>, where *P* is a stage of a process (i.e. a Refal expression with possible activation brackets but without variables), produces ↑*P'*, where *P'* is the next stage of the process *P*, i.e. the expression to be found in the view-field of the Refal machine after one step is applied to *P*.

The usual mathematical notation is semi-formal, i.e. it can be converted to formal objects but the conversion process is not defined in a formal language. For the sake of mathematical discussion, we shall use in the following a semi-formal functional notation, in which the process initiated by a machine *F* with inputs *X*, *Y*, etc., i.e. something which could look in Refal as <*F*(*X*)(*Y*)...>, will be denoted as *F*(*X*,*Y*, ...). Metacode transformation is not reflected in the semi-formal notation, so that both <F<G ...>> and <F *(G ...)) are denoted as F(G(...)). To translate the semi-formal notation into the strict Refal notation, one must know what functional arguments are called as values (unmetacoded), and what as processes (metacoded); this will be usually clear from the context.

4. Models, Predictions, Propositions

We proceed now to examine the intuitive notion of a *model*. As stated in Introduction, a mathematical proposition has a meaning to the extent it produces some models of reality. Now we want to formalize the notion of a model and find something like minimal units of semantics, some elementary propositions, such that combining them we could construct every meaningful proposition.

Informally, we say that the process *B models* the process *A* if there is some similarity between the stages of *B* and *A*. It is not necessary that every stage of *A* or *B* be related to some stage of the other process; generally, we select some stages in *B* which should be somehow put into correspondence with some stages selected from *A*. Let the selected stages of *A* be A_1, A_2, etc.; the corresponding stages of *B* will be B_1, B_2, etc. The statement that

B models A is then composed of the statements that B_i is "similar" to A_i for $i=1,2$, etc. We call these statements *predictions.* This corresponds to our intuitive notion of a prediction. Indeed, when we predict that at a certain time a certain planet will have certain coordinates, we specify: (1) a certain stage A_l of the natural process of the movement of celestial bodies, (2) the final stage B_l of the linguistic process B of astronomical calculations, and (3) a relation of "similarity" between B_l and A_l which is tested by the process of coordinate measurement.

To formalize the notion of prediction we assume, in the spirit of our philosophy, that all the three components indicated above can be represented by certain exactly defined processes. Then their combination is also a process: the one that verifies that A_l and B_l do exist, and that the process of testing their "similarity" comes to a successful end. Therefore, we define a prediction, generally, as a statement that a certain process (search) is finite.

In mathematics, the process A, which is the object of study, is linguistic, as well as the process B; therefore the process of testing predictions can be defined by a linguistic machine. Consequently, the semantics of mathematics can be completely formalized within the linguistic sub-universe.

Note that we made the weakest possible assumption about the testing process. We do not assume that it always stops and answers 'yes' or 'no' to the question of whether the arguments are in a given relation; it does not implement a total recursive predicate. Our testing process implements what may be called a *semi-predicate*: it can say 'yes', but instead of saying 'no', it simply goes on and on without ever stopping. Using only testing machines we can express everything that can be expressed through total recursive predicates. Indeed, suppose we want to imitate a machine $\langle\sigma(e_x)e_y\rangle$ which always stops and produces T or F as the answer. We can construct a testing machine:

$$\langle\rho_t(e_x)e_y\rangle \rightarrow \langle\underline{\text{loopf}}\ \langle\sigma(e_x)e_y\rangle\rangle$$
$$\langle\underline{\text{loopf}}\ T\rangle \rightarrow$$

<loopf F> → <loopf F>

which stops if and only if σ produces T. Analogously we define a testing machine ρ_f which stops if and only if σ produces F. Running the two testing machines in parallel will allow us to do anything that can be done by running σ. The converse statement, that whatever can be expressed through semi-predicates can also be expressed through total recursive predicates, would not be true, of course. The semi-predicate is a smaller semantic unit than the recursive predicate: "one half" of it.

If a process is represented in Refal by *P*, the prediction that it is finite will be represented by ↑*P*!, i.e. the metacode of *P* followed by '!'. We need a metacode transformation here in order to be able to deal with predictions as objects.

A model of a process may yield a finite $\overset{or}{\text{of}}$ infinite number of predictions. Thus, we formalize the notion of a *model as a generator of predictions*. To deal with models as with objects, we have again, as in the case of predictions, to use metacode. So we define in Refal a generator *G* of predictions and take ↑*G* as the linguistic representation of the corresponding model.

A couple of examples. The statement that a process <αAAA> is finite is the prediction *(αAAA)!. If the α-machine is defined as in Sec.2, then this is a false prediction, because this process is infinite. The process <+(01)011> is finite. The corresponding prediction, *(+(01)011)! , is true.

Our treatment of mathematical statements is close to the one accepted by intuitionists. We read in [Heyting 1966]:

"... a mathematical theorem expresses a purely empirical fact, namely the success of a certain construction. '2+2=3+1' must be read as an abbreviation for the statement: "I have effected the mental constructions indicated by '2+2' and by '3+1' and I have found that they lead to the same result."

We move still further in the direction of empiricism and replace mental construction by a linguistic one. To express Heyting's arithmetic statement, we use the function of addition as defined above, and define a tester of equality = , such that $\langle=(e_1)(e_2)\rangle$ stops if and only if e_1 and e_2 are identical numbers:

$$\langle=(\emptyset)(\emptyset)\rangle \rightarrow$$
$$\langle=(e_x1)(e_y1)\rangle \rightarrow \langle=(e_x)(e_y)\rangle$$
$$\langle=e_x\rangle \rightarrow \langle=e_x\rangle$$

Now our arithmetic proposition is:

$$\uparrow\langle=(\langle+(\emptyset 11)\emptyset 11\rangle)(\langle+(\emptyset 111)(\emptyset 1)\rangle)\rangle!$$

Consider the proposition:

$$(Ax)(x+\emptyset = x)$$

where quantification is over all whole numbers. This is a generator which produces predictions

$$\emptyset+\emptyset = \emptyset$$
$$1+\emptyset = 1$$
$$2+\emptyset = 2$$
$$\ldots$$

etc. It is not difficult to define such a generator in Refal.

An existentially quantified formula like

$$(Ex)(5+x = 8)$$

will be interpreted in our theory as the finiteness of the search for the value of x satisfying the equation.

Let us now consider the statement that a given process is infinite. What is its meaning? Can it be understood as a generator of predictions?

Yes, to state that a process A is infinite is to state that:

• the initial stage A is not passive (includes at least one pair of activation brackets);
• the next stage after the initial stage is not passive;
• the next stage after the next stage after the initial stage is not passive, and so on, infinitely.

Every one of these statements can be formalized as a prediction by defining a process which checks whether a given expression is not passive, and stating that this process when applied to a given stage of A is finite. Thus the infinity of a certain process is an infinite generator of predictions.

Because of the importance and frequent use of the infinity model we introduce a special notation for it. The proposition that a process A is infinite will be represented by the object expression ↑A?, and in the following we shall treat such propositions, together with predictions, as certain elementary units, *atoms*. Thus, propositions ↑A! and ↑A? will be called *atomic*. One should bear in mind, however, that while ↑A! is a prediction, ↑A? is a *generator* of predictions, which can also be written as *(inf ↑↑A) with a properly defined generator inf.

We came close to a general definition of proposition. We have only to make one last generalization. It is not necessary that a generator produce only predictions ready for use. It may also produce generators, which in their turn produce predictions; and generators, which produce generators which predictions, etc. We shall say that *P hierarchically produces Q*, if there is a finite chain $P_1, P_2, \ldots P_n$, such that P_i produces P_{i+1}, $P_1=P$, $P_n=Q$. We come to the following inductive definition:

(a) a prediction is a proposition;
(b) the metacode of a generator which generates only propositions is a proposition.

Thus a proposition may produce a whole hierarchy of propositions, but they must be such that *ultimately* they produce *predictions*. A formal object has a meaning as a proposition only to the extent we know how to make it produce predictions. If there is no way to obtain predictions from an object, it has no meaning as a proposition. Atomic propositions constitute the ground level of the hierarchy of propositions. We recognize them syntactically by the fact that they end with a symbol '!' or '?'. If a proposition does not end with one of these it should be treated as the metacode of a machine which can still be run to produce lower-level propositions.

5. Logical Connectives and Quantifiers

Now we are going to interpret the means logic has for the construction of composite propositions: connectives and quantifiers.

Let us start with conjunction. To uphold two or more propositions means, obviously, to uphold all the predictions produced by any of them. So we define the function and with the format <and *L*>, where *L* is a list of propositions:

$$\langle \underline{\text{and}}\ (e_1)e_2\rangle \rightarrow (e_1)\ \langle \underline{\text{and}}\ e_2\rangle$$
$$\langle \underline{\text{and}}\ \rangle \rightarrow$$

If P_1, P_2, ... ,P_n are propositions, then the process

$$\langle \underline{\text{and}}\ (P_1)(P_2)\ldots(P_n)\rangle$$

will generate all of them and only them. Its metacode

$$*(\underline{\text{and}}\ (\uparrow P_1)(\uparrow P_2)\ldots(\uparrow P_n))$$

is our formalization of the conjunction. In semi-formal notation we write: $\underline{\text{and}}(P_1,\ \ldots\ P_n)$.

To formalize disjunction, consider first the case when both operands are predictions: S_1! and S_2!. The disjunction of these propositions is the statement that at least one of the two searches S_1 and S_2 is finite. This is the same as to say that the process in which S_1 and S_2 are run in parallel is finite. So we construct the following machine:

$$\langle \underline{or}(e_1)(e_2)\rangle \rightarrow s\left|\begin{array}{l}\langle \underline{act}\ e_1\rangle\\ \\ \langle \underline{act}\ e_2\rangle\end{array}\right.$$

where function act "activates" a metacoded expression, i.e. simulates the corresponding process.

Now the disjunction $S_1!$ or $S_2!$ is formalized as the finiteness of the process $\langle \underline{or}(S_1)(S_2)\rangle$, that is the prediction $\ast(\underline{or}(\uparrow S_1)(\uparrow S_2))!$. In semi-formal notation, $\underline{or}(S_1,S_2)!$.

Let the operands of a disjunction be the general propositions P_1 and P_2, i.e. prediction generators, not just predictions. How do we then interpret the disjunction?

We face here a new situation. Until now we were able to interpret every mathematical proposition in terms of prediction generators. But the disjunction P_1 or P_2 cannot be immediately interpreted in this way. We can state neither of the predictions generated by P_1 or P_2, because every one of them may be false. This does not mean, however, that P_1 or P_2 is meaningless. It is simply that this statement includes a new factor, namely the concept of truth, which was not present in the meaning of those statements we considered before. Note that even putting the meaning of the disjunction in words we cannot avoid a reference to the concept of truth; we say: "at least one of P_1 and P_2 is true".

But what is the meaning of the statement that a given proposition is true? No matter how we answer this question in the plane of philosophy, the statement that P is true means, in the empirical plane, that some processing was applied to P and gave a positive answer, or, in terms of our theory, a search for justification of P as true has turned finite. We shall denote this search as $\langle \gamma\ P\rangle$. Accordingly, the statement that P is true is the prediction $\uparrow\langle \gamma\ P\rangle!$, or just $\gamma(P)!$ in semi-formal mathematical notation.

The concept of the γ-process underscores the empirical character of our approach. We refuse to accept any concept of truth (the classical correspondence concept, in the first place) if it cannot be expressed in terms of predictions about some actual processes. We declared such concepts meaningless. Hence if we are to use the concept of truth (and we saw that we could not avoid it in interpreting logical connectives), we must introduce explicitely a truth-testing process, which is implicite in the meaning of logical formulas. The nature of this process will be our major concern in the rest of this paper. In this section we shall simply use the notation $\gamma(P)$ for a further look into logical connectives.

However, it should be clear already that the introduction of the truth-testing process brings in an element of subjectivity into the theory. A prediction A!, where A is a deterministic mechanical process, can be verified in an undisputable way by simply running the process A and seeing that it has stopped. The statement A! is defined *objectively*. An infinite generator of predictions, in particular A?, cannot be objectively verified. Any truth-tester for such a proposition has to rely on our knowledge and intuition, which may vary from one subject of knowledge (user of theory) to another.

Our formalization of the disjunction P_1 or P_2 is or$(\gamma(P_1),\gamma(P_2))$!. This is a prediction, but one about a user-dependent process. The inroduction of this process is not our whim, but a shear necessity. A reference to this process has been there before: it is in the meaning of logical propositions.

Our semantic definition of proposition leads to a natural interpretation of *logical implication*. A proposition P_1 *logically implies* a proposition P_2 if P_2 is among the propositions generated by P_1. This definition is the most exact formalization of the intuitive concept of logical implication, according to which if P_2 is implied by P_1, it is already somehow contained by P_1; included in it. We can define in Refal the machine imp with the format <imp$(e_p) \rightarrow e_q$>, which tests whether e_q is one of the propositions produced by e_p. The proposition that P logically implies Q is then, in semi-formal notation, imp$(P \rightarrow Q)$!.

The relation between P and Q expressed by imp$(P \rightarrow Q)$! is very different from the *material implication* of mathematical logic. The latter, unlike the former, can connect two arbitrary propositions which in no way are related by their meaning. It establishes the connection by force, so to say, announcing it as an empirical fact, a new law of nature. Using '$\rightarrow$' to symbolize implication, we can declare that

$$(x \text{ is an apple}) \rightarrow (x \text{ is on the table})$$

even though the definition of the concept of apple does not include that it is necessarily put on the table. Compare this with the following implication:

$$(x \text{ is an apple}) \rightarrow (x \text{ is a fruit})$$

This proposition, like the preceding one, can be put foreward as a material implication, but it is also true as a logical implication, because being a fruit is a part of being an apple. In Kant's terminology, logical implication forms an *analytic* judgement, while material implication forms a *synthetic* judgement.

In our theory we formalize both logical and material implication and one can see how different these concepts are. In contrast, the conventional mathematical logic has only one implication: material. The closest thing to logical implication that mathematical logic has is the concept of deducibility: Q is deducible from P if the (material) implication $P \rightarrow Q$ is a tautology. The difference between the two kinds of implications becomes here a meta-concept referring to the way we deal with propositions, not a property of the propositions themselves, as we conceive it intuitively. This reflects, of course, the purely syntactic (formal) nature of mathematical (formal) logic, and constitutes, in our view, its main deficiency. Formal logic has nothing to do with the meaning of the constructs it introduces. For instance, when the connective and is defined, it is nowhere to be seen that P and Q *logically implies P*, so we have to state it as a *material* implication. In our theory, the definition of the and connective formalizes our intuitive understanding of it, its *meaning*. Accordingly, we do not have to postulate that *P* and Q logically implies *P*, we can prove it.

We shall use the notation

if *P* then *Q*

for the material implication involving a pair *P* and Q, where *P* is the *antecedent*, and *Q* the *consequent*. How can we formalize this concept?

Consider first the case where the antecedent of an implication is a prediction: 'if the process *A* is finite then proposition *P*'. There is an obvious way to interpret this proposition as a generator of predictions: we run the search *A*, and when/if it stops, produce proposition *P*. We define the if machine as follows:

$$\langle \underline{\text{if}}(e_a)!\underline{\text{then}}\ e_p\rangle \rightarrow \langle \underline{\text{second}}(\langle \underline{\text{act}}\ e_a\rangle)(e_p)\rangle$$
$$\langle \underline{\text{second}}(e_1)(e_2)\rangle \rightarrow (e_2)$$

This machine activates the process e_a, and when and if it ends, generates e_p . The auxiliary function second is needed to discard the first element of a list and output the second element. The parentheses in the final result are necessary because a proposition-generator, according to our convention, produces a *list* of propositions (which in this case consists of one member). If the process e_a is infinite, the if machine will go on infinitely, producing nothing.

So, if ↑*A*! is a prediction and Q an arbitrary proposition, then the material implication of the latter by the former is:

↑<if(↑*A*)!then Q> ,   or semi-formally: if*A*!thenQ

Now let the antecedent be a prediction generator, for instance an infinity model. What do we mean when we say "if the process *A* is infinite then proposition *P*"?

As in the case of disjunction, there is a hidden reference here to a process which establishes the infiniteness of *A*. What we actually mean is "if *we can know that A* is infinite then *P*". Therefore, for an arbitrary proposition *P*, the material implication $P \rightarrow Q$ is interpreted as:

<u>if</u> $\tau(P)$! <u>then</u> Q

The property of implication that a false antecedent forms a true proposition with any consequent shocks everyone who studies mathematical logic for the first time as contradicting our intuition and common sense. Then one gets used to it and accepts the usual justification, namely that taking such a proposition as true we can derive a false proposition (by the *Modus Ponens* rule) only if we have already derived at least one false proposition, the antecedent; but then our theory is already false, so we do not care. Here we clearly see the contradiction between the purely syntactical, *asemantic* nature of the conventional mathematical logic and our unexpressed expectation that a formal logic will pick up and codify the essence of different forms of thought, which is their *meaning*. In our theory, an implication with a false premise is not true in the same sense as a prediction, or a generator producing true predictions can be true. Neither is it false. It is *empty*: a generator which produces nothing. This, we believe, is in perfect agreement with our intuitive expectation.

To formalize universal quantification, we define function <u>all</u> with the format <u>all</u>($V \in G$: *P*), (from now on, we shall use only semi-formal notation), where V is a variable, *G* is a set generator, and *P* is a propositional form depending on *V* (or any expression which may include *V*). The <u>all</u> machine activates the generator *G* step by step, substitutes the objects produced by *G* for every *V* in *P*, and produces the resulting expression. For instance, if <N 0> is the generator of all natural numbers, and *P*(*x*) is a propositional form representing in our theory some predicate P(x) of formal logic, then

$$\underline{\text{all}}(x \in \mathrm{N}: P(x))$$

is our representation of (Ax)P(x), where x runs over all natural numbers.

To express existential quantification we define the searching machine <u>sch</u> with the format <u>sch</u>($V \in G$: S), where V and G have the same meaning as above, and S is a search which may depend on the variable V. The <u>sch</u> machine runs the generator G, substitutes the produced objects for V in S and runs all the resulting searches S in parallel. The moment any of these searches comes to an end, the <u>sch</u> machine also stops and outputs as its result the object which has made S finite.

If a predicate P(x) of formal logic can be represented as P! in our theory, the quantified proposition (Ex)P(x) will be represented as <u>sch</u>($x \in G$: $P(x)$)!. If P(x) corresponds to a general proposition (generator) P, then we must use function γ to convert it into a search: <u>sch</u>($x \in G$: $\gamma(P(x))$)!.

Negation in our theory, as in intuitionism, is the most sophisticated of the connectives. We introduce a special device to establish the falsity of P. The negation of P is formalized as as $\bar{\gamma}(P)$! To define the $\bar{\gamma}$-process, we must concretize our concept of the γ-process.

The γ- and $\bar{\gamma}$-processes are related through the concept of *human knowledge*

We define a *knowledge* as a proposition which is believed to be true. This definition reflects our subjective attitude towards knowledge and the way we use it. We use propositions believed to be true in order to make predictions, and we believe in these predictions, i.e. plan our actions under the assumption that actual processes of the world will conform to the predictions. The question whether what we call our knowledge is actually true is left to an observer (if any) who watches us from outside. The concept of truth is inseparable from an observer, like some fundamental concepts of modern physics. The truth of a proposition is somebody's readiness to plan his actions in accordance with the predictions implied by it. In the absence of this 'somebody', the concept of truth becomes meaningless.

If a proposition is a generator producing predictions infinitely, we cannot verify them all. Our readiness to rely on it as true is based, in the last analysis, on a belief, and not on an empirically established truth. We do not discuss at this time the philosophical question of how this belief is arrived at; the only thing we can say about our knowledge is that "we believe because we believe". Then the only thing we can do in our theory is simply introduce a symbol to denote the sum total of mathematical propositions believed to be true by the subject of knowledge. We shall use the capital Greek letter Γ (for 'gnosis', knowledge) as such a symbol. Now we can define the γ-process as follows:

$$\langle \gamma e_P \rangle \rightarrow \langle \underline{imp}(\Gamma) \rightarrow e_P \rangle$$

The γ-machine tries to deduce its argument from the knowledge Γ. If $\langle \gamma P \rangle$, where P is a proposition, is finite, then P is true because it is implied by our current knowledge. If it is infinite, then we can say nothing.

Using the concept of human knowledge Γ, we can define negation as contradiction to Γ.

Atomic propositions $A!$ and $A?$ with the same search A will be called *opposite*. A pair of opposite atomic propositions is a *contradiction*. A proposition is *contradictory*, or *inconsistent*, if it produces a contradiction. Otherwise it is *consistent*.

We can construct a machine which tests that a given proposition is contradictory. Let its format be $\langle \underline{con}\ e_P \rangle$. It runs e_P and keeps track of all the atomic propositions produced. Whenever a new atomic proposition is produced, <u>con</u> examines it against the all those produced previously. If a contradiction is discovered, <u>con</u> stops, otherwise it goes on infinitely.

If P is a proposition, we interpret its negation as the statement that the conjunction of P and the human knowledge Γ is contradictory: $\underline{con}(\underline{and}(\Gamma,P))!$. Defining function $\bar{\gamma}$ as

$\bar{\gamma}(P) \rightarrow$ con(and(Γ,P))

we represent the negation of P as the prediction $\bar{\gamma}(P)$!.

So, we have two "cognitive" functions: $\gamma(P)$ establishes the truth of proposition P, $\bar{\gamma}(P)$ establishes its falseness.

By introducing a notation for human knowledge we do nót solve all our problems, however. The symbol Γ is a *metasymbol*; it stands for some expression which we do not write out explicitely. But here is a problem: the human knowledge -- be it that of one person, or of the human race as a whole -- does not stay the same; it is developing, growing. Essentially, it is a process, not just an object expression. Then how shall we interpret the concept of truth with respect to this ever changing knowledge?

Two answers to this question are possible, both consistent if kept firmly to. As we shall see later, the first answer leads to the intuitionist logic, while the second to the classical.

Intuitionist logic. Since the meaning of propositions depends on Γ, we consider the meaning definite only if a definite Γ is indicated. We can think of Γ as the sum total of human knowledge at the present time. Therefore, $\langle\gamma P\rangle$ will be finite and P accepted as true, only if we actually performed the proof process based on a definite Γ. Although Γ changes as the human knowledge is growing, at any particular moment in time Γ should be treated as a definite fixed expression.

Classical logic. When we speak, e.g., of existential quantification, we do not say "such an x that we can prove $P(x)$", we say "such an x that $P(x)$ is *actually* true", even though we may not be able to find this x on the basis of our present knowledge. Thus we refer not to our present knowledge, but to an imagined *complete* knowledge, which implies all the propositions that we may find true now or at any future time. From Goedel's theorem we know that no definite expression Γ can represent this complete knowledge. We can see Γ only as an unreachable limit of the expanding human knowledge, or as the expanding human knowledge itself (a process), because its every stage includes the knowledge which existed at all the past stages (a cumulative process).

It is pretty clear that a consistent theory can be based on the intuitionist viewpoint. Our main effort will be to show how classical logic and set theory based on it work, and to prove their consistency.

6. The Concept of Truth

Intuitionism is based on a philosophy different from that of classical mathematics. In our theory, the difference manifests itself in the definition of the truth-seeking processes. But it may seem that even within one philosophy of mathematics we have two different concepts of truth: (1) A proposition is true iff it hierarchically generates only true predictions, and (2) A proposition P is true iff the process $\gamma(P)$ comes to a halt. In fact, however, we have a single definition of truth, of which (1) is the intuitive, informal version, while (2) is its formalization in our theory.

To better understand this situation, let us draw a parallel with physics. Consider a particle. The notion of its *position* in the space is intuitively quite clear and unambiguous. Should there be no such concept as *coordinates*, one would never think that the notion of the particle's position is flawed or insufficient. Yet to turn this intuitive notion into an exact physical concept, we use coordinates, and the process of measurement which yields them.

In our theory we treat the intuitive notion of the truth-value as the physicist treats the notion of position. We define processes which determine the truth-value of every proposition, *measure* it, in a sense. Like the process of measurement in physics, the process of truth-seeking is started by a decision of the subject of knowledge and may vary considerably, yet it is not completely arbitrary but must conform to certain requirements, or follow certain rules. We set two rules, which are, essentially, the definition of the process of mathematical cognition represented by the metasymbol Γ. The *consistency rule* says that we accept a stage Γ_i of Γ as knowledge only if we cannot produce any contradictions from it. The *completeness rule* demands that whenever we discover a proposition P such that $\underline{\text{add}}(\Gamma_i, P)$ produces, to the best of our knowledge, no contradictions, yet P is not produced by Γ_i, we add P to the knowledge, i.e. take $\Gamma_{i+1} = \underline{\text{add}}(\Gamma_i, P)$ as the next stage of Γ.

The original intuitive definition (1) is equivalent to (2). By the consistency rule, whenever $\gamma(P)$ is finite, P produces only true predictions. By the completeness rule, whenever P produces only true predictions, $\gamma(P)$ is finite.

But is it not possible that some P is, in actual fact, true, but $\gamma(P)$ is infinite? No, it is not. Because the meaning of the phrase "P is, in actual fact, true" is that it is possible to prove in a convincing way that P is true, if not with out current knowledge Γ_i, then at a later stage. Otherwise the phrase makes no sense (except, maybe, a metaphysical one, which is not our concern here).

Mathematics, as every other branch of science, makes progress by the method of trial and error, which is known to mathematicians as *reductio ad absurdum*. Whenever we state the current Γ_i as our knowledge, we cannot, theoretically speaking, be completely sure that at the next stage we shall not find Γ_i inconsistent; our acceptance of certain powerful propositions as true is a mere consensus. Yet in our theory we assume that every stage Γ_i of the cognition process is true. Does it not invalidate the theory?

As every scientific concept, the concept of the cognitive process is *idealized*. A parallel with physics will help again. Particle coordinates, as they are used in theory (we are speaking of classical, not quantum, mechanics), result from idealized incorruptible measurements, which can never be realized in actual fact. The discrepancy between actual and idealized measurements, i.e. the errors of measurments, are taken care of, but at a different level, not in the theory itself. The rational members

x_i of a Cauchy series representing a real number x can be seen as the result of an idealized measurement; they can be compared with the stages Γ_i of the idealized cognitive process Γ. An error made by a human being, or a computer, in a proof, has no more (and no less!) effect on our theory than an erroneous measurement has on the theorems of classical mechanics.

The method of getting mathematical knowledge exemplifies the general method of science. We put foreward a hypotetical general statement, a generator of predictions P, add it tentatively to our knowledge, and test against the verifiable facts, predictions. If there are no contradictions, P stays in the knowledge, although, in principle, as a hypothesis only. Recognizing this process as the essence and the meaning of the concept of truth in mathematics, we formalize it and put it in the foundation of mathematics, like the physicist who formalizes the process of measurement as the concept of real number. Our empiricist approach stands in contrast to the traditional, if not always articulated, idea of mathematics as having a subject and a method which is radically different from those of natural sciences. This latter idea has lead people to define the concept of a mathematical truth as something soaring in high spheres of one kind or another (Plato, of course, was the first who expressed this idea in its purest form), and think about truth-testing as something secondary, derivative. We reverse this relationship, following the lead of physics, which achieved spectacular successes by putting observable facts and processes in front of metaphysical images which uncriticised intuition takes for real entities.

To identify the concept of truth with the cognitive process expressed by Γ, γ and $\bar{\gamma}$, we have made two idealizing assumptions about this process: its consistency and completeness. Since the cognitive process is a creative process freely initiated by the subject of knowledge -- i.e. by the party referred to as "we" and "us" in mathematical texts, these assumptions should also be described as the rules we have chosen to adhere to in order to discover mathematical truths. By the consistency rule, when we see that P is false we cannot add it to the knowledge; the infallibility of the idealized user is an expression of this self-correcting nature of the cognitive process. By the completeness rule, if we are convinced that *P* is true we cannot leave it alone and *not* to add to the knowledge.

7. Real Time and Model Time

Mathematics is the art of constructing the most fundamental models of reality. We formalize mathematics using the concept of the Refal machine. To create models of reality by means of the Refal machine we: (1) put into its program field some definitions, (2) put into its view-field some 'processes' (which are, in fact, certain expressions representing initial stages of processes) and start the machine. The processes going on in the view-field of the Refal machine are modelling real world processes.

We can distinguish two time scales here; two "times" as it were. We write definitions and put them into the memory of the Refal machine as living human beings, *in real time*. When we run the Refal machine, the sequence of its steps represents another time: the time of the mechanical process we initialized. Although the process to be modeled occurs, presumably, in real time again, the Refal machine runs in a different time, which we shall refer to as *model time*. We can compress or expand model time unlimitedly -- in imagination if not in reality. We can run the Refal machine at a speed of one step, or one thousand steps, or one million steps per second. No matter what the actual speed is, we still can imagine a speed that is twice as high. Moreover, we can examine the stages of a mechanical process in the inverse order, that is we can reverse model time. Model time, unlike real time, is completely subject to our will. Model time is a feature of the machines we run using hardware or imagination.

Knowledge is the existence in a cybernetic system of a model of some part of reality. Knowledge is both objective and subjective because it results from the interaction of the subject and the object of knowledge. We know that knowledge is never complete. But if the information flow between the subject and the object is only in one direction -- from the object to the subject, we can imagine a complete knowledge (i.e. a complete perfect model) of the object; this idea is not contradictory. If there is a flow of information from the subject to the object too, the notion of a complete model may become contradictory. Let S_1 be a system-object, and S_2 a system-subject. Suppose S_2 includes a complete model of S_1. Then it can run this model, compressing model time with respect to real time, and predict the behaviour of S_1 for all times in advance. This prediction can be sent to S_1, which can change its behaviour so as to falsify the prediction. This contradicts the assumption of a complete model. In particular, the <u>notion of complete self-knowledge</u> (the case when $S_2 = S_1$) <u>is contradictory</u>.

It follows from this reasoning that when we are dealing with the processes of self-knowledge we must clearly distinguish between real-time processes and model-time processes. If we allow the difference to be blurred we may come to absurdities. For example, when we say "imagine a real-time process *A*" we already are in a danger zone, because our imagination creates a model-time process which is not the same as the original real-time process. It will cause no trouble if process *A* is detached from ourselves (*we* is the subject of knowledge), so that it cannot be influenced by what we are doing; otherwise, we can only say "imagine a *model* of a real-time process *A* , which, of course, is only partial".

One can picture a real-time process as consisting of a definite past and indefinite future, with a border between them known as "this moment", or "now". A real-time process, generally, is not deterministic. Any statement concerning its future is in the mode of possibility, not necessity. When we say that the process $\gamma(P)$ *is* finite, we mean that it *can be* finite. Even when the subject of mathematical knowledge is the whole of humanity, there is no guarantee that for every true (i.e. producing only true predictions) proposition *P* the process of proving *P* will be actually completed. If humanity exterminates itself, or is destroyed by a cosmic catastrophe, a lot of theorems will be left unproved.

Our theory is a formalization of the self-knowledge of mathematics. Therefore the distinction between real-time processes and model-time processes is for us an absolute necessity. So, how should the difference between real time and model time enter our theory?

The processes that take place in the view-field of the Refal machine as the result of the application of sentences are *model-time* processes. Should the program field be fixed once and for ever, there would be only model-time processes in existence, and no change in real time. But in fact it is not fixed at all. The program field of the Refal machine which represents mathematics is changing in real time as we create more and more mathematics, which means that we define new processes and expand our knowledge of the processes already defined.

It would be exceedingly inconvenient if we had to consider every part of the memory (program field) of the Refal machine as potentially variable in real time. We could hardly come to any definite conclusions in such circumstances. But we can define a certain number of real-time processes and give them a place in our formal system. Air temperature in the City of New York could be such a process. Or the mathematical knowledge of mankind.

We shall represent real-time processes as follows. Like model-time processes, a real-time process is distinguished from an object by a pair of activation brackets which delimit its representation. The contents of the brackets may be anything which identifies the process, e.g. a symbol. However, the activation of these brackets does not bring about all stages of the process, as in the case of a model-time process (we do not know them all in advance), but produces (in one step) only the current stage of the process. Thus real time remains real time and no attempt (not even a concealed one) is made to substitute a model-time process for a real-time process. The notation <*R*> of a real-time process is a device which allows the Refal machine to have access to the process; which means to its current stage. We can describe this situation as the presence in the program field of a sentence:

$\langle R \rangle \rightarrow E$

where E is an object expression which represents the current stage of the process $\langle R \rangle$ and changes in real time. For instance, we can use <tempNYC> in a Refal program and declare that this term will be replaced, as the Refal machine works, by the current temperature in New York expressed in agreed units with an agreed precision. The results may be different if we run such a program today and tomorrow.

Our formalization demonstrates the profound difference between real time and model time, which was first realized by Henri Bergson, who described real time as "duration" and contrasted it with the time as known in mechanics, which he saw as a projection of duration on space. Norbert Wiener, in his *Cybernetics*, also juxtaposes Newtonian and Bergsonian time. In our terms, Newtonian time is the model time: a feature of deterministic mathematical models of reality. Bergsonian duration is represented in our theory by real-time access functions: this is something very different from autonomous machinery representing model time.

The Refal machine is a formalized version of a part of the cognitive apparatus of the human being. We can compare this apparatus with a complex computer system. In computer systems we distinguish subsystems which work *off-line* and those which work *on-line*, in real time. Running off-line subsystems is analogous to those processes in human brain which we describe as imagination. Usually we see mathematics as dealing with our imagination. This is generally true, but with a notable exception: when developing metamathematics, i.e. mathematical self-knowledge, we cannot, as disussed above, limit ourselves to model-time processes only, because human knowledge is a real-time process which has no complete model. Thus the model of human cognition which we are constructing must recognize the fact that our brains do not exist in our imagination only, but are real cybernetic systems which exist in real time and have subsystems which work on-line, in real time.

Access functions to real-time processes imitate, or model, links between our cognitive apparatus and the real world; they are sort of 'sense organs' of the Refal machine. The real world, it must be noted, includes not only things external to us but also the current state of our cognition. We can create models of external processes, and then models of our models of external processes, and models of models of models, etc., but the whole hierarchy will invariably exist in the real world and will be open to change in real time.

The knowledge Γ which shows up in the semantics of mathematical propositions is a real-time process. To connect the Refal machine with this process we use the access function <gns> ('gnosis'). The symbol gns is a regular Refal symbol. The activation of gns gives (in one step) a proposition which sums up all the knowledge we (the subject of mathematical knowledge) have at the present time.

We shall use subscripted symbols Γ_1, Γ_2, Γ_i, etc. to denote specific stages of the human knowledge process. Since the result of the activation of <gns> may be different at different times, the process <gns> is *undefined* in Refal, or, equivalently, defined by a sentence

$$\langle \underline{gns} \rangle \rightarrow \Gamma$$

the right side of which is provided by the user. The cognitive functions γ and $\bar{\gamma}$ represent the processes of testing truth and falsehood. As we mentioned before, our formalism allows two interpretations of the cognitive functions. One interpretation treats Γ as a definite expression and leads to intuitionist logic. This interpretation is *static* with respect to the real-time process of human knowledge. It does not exclude the possibility of Γ changing in real time, but during one run of the function γ (or $\bar{\gamma}$) the stage of Γ is taken to be fixed, unchanging. The following sentences define the cognitive functions in static interpretation:

$$\gamma(P) \rightarrow \underline{\text{imp}}(\underline{\text{gns}} \rightarrow P)$$
$$\bar{\gamma}(P) \rightarrow \underline{\text{con}}(\underline{\text{and}}(\underline{\text{gns}},P))$$

These functions are not autonomous machines; they depend on real time. But once the concretization of <$\underline{\text{gns}}$> is done, the further operation of $\gamma/\bar{\gamma}$ in static interpretation is autonomous, mechanical. Essentially, we deal here not with one concept of truth, but with as many concepts as many stages of Γ_i or Γ are there.

The other interpretation corresponds to our intuition of *objective* truth and leads to the usual, classic logic. In terms of our theory it is *dynamic*, because it takes the real-time process Γ as a whole and involves all stages Γ_i of Γ in the processes γ and $\bar{\gamma}$. The process $\gamma(P)$ is running in parallel the searches $\underline{\text{imp}}(\Gamma_i \rightarrow P)$ with each new stage Γ_i as it appears in real time, and $\bar{\gamma}(P)$ is defined analogously:

$$\gamma(P) \rightarrow s\left|\begin{array}{l}\underline{\text{imp}}(\underline{\text{gns}} \rightarrow P)\\ \\ \gamma(P)\end{array}\right.$$

$$\bar{\gamma}(P) \rightarrow s\left|\begin{array}{l}\underline{\text{con}}(\underline{\text{and}}(\underline{\text{gns}},P))\\ \\ \bar{\gamma}(P)\end{array}\right.$$

In dynamic interpretation, $\gamma(P)$ and $\bar{\gamma}(P)$ are genuine real-time processes in which the operation of the Refal machine is intertwined with the process of human knowledge. They depend on the time interval Δt that it takes for the Refal machine to make one step, but those features of the cognitive functions which are essential for our theory do not depend on Δt, as we shall discuss in more detail in Part 2. Let Γ_i be the state of human knowledge at the moment when the Refal machine is making its i-th step. Then the function $\gamma(P)$ will run in parallel the searches imp($\Gamma_1 \to P$), imp($\Gamma_2 \to P$), etc., each next process one step behind the preceding. If a stage Γ_i such that imp($\Gamma_i \to P$) is finite ever appears in human knowledge, and only in this case, the search $\gamma(P)$ will be finite. The search $\bar{\gamma}(P)$ will be finite if and only if P contradicts some stage Γ_i.

We now have two kinds of processes:

(1) Those which never call real-time access functions. We shall call such processes *mechanical.* They are generated by an autonomous machine and are completely defined and deterministic.

(2) Those which at one stage or another call a real-time access function. A machine that generates such processes is not autonomous, it is open to the user, the subject of knowledge. We assume from now on that the Refal machine can have no direct contact with physical processes in the world bypassing our (the user's) consciousness; whatever is changed in the memory field is changed by our decision. Processes generated by such interaction between the subject of knowledge and the machine will be called *metamechanical*. In terms of computer science, the machine here works in the *interactive mode*, and the subject of knowledge is the user of this machine. The 20th century's physics has discovered that we cannot eliminate the subject of knowledge from our picture of the physical world. Our theory reflects an analogous situation in mathematics. Mathematical knowledge is the construction of machines to model reality, but these machines do not always work autonomously: some are used in the interactive mode.

Combination of logical connectives may lead to predictions about processes which ~~which~~ call cognitive functions, i.e. predictions about metamechanical processes. For instance, the law of the excluded middle is or($\gamma(P)$,$\bar{\gamma}(P)$)!. But how are we to judge whether such a statement is true or false? A prediction about a mechanical process is well defined because the process itself is well defined and deterministic. But metamechanical processes are not deterministic, so predictions about them have no meaning! At least, not before we somehow define their meaning.

In the second part of this paper we shall give an interpretati to statements about metamechanical processes of mathematics which closely matches our intuitive understanding of objective truth. But to admit this interpretation, a metamechanical process must meet certain requirements; we call such processes and propositions about them *objectively interpretable*. Those proposition which are not objectively interpretable have no meaning. It is by using these meaningless propositions that people come to the well-known logical and set-theoretical paradoxes.

Acknowledgements

I appreciate discussions of the Cybernetic Foundation which I had with several colleagues: Karel Hrbacek and Michael Anshel of the City College, Angus Macintyre of Yale, and Avgusin Tuzhilin of the College of Staten Island.

REFERENCES

Bergson, Henri. *Time and Free Will*.

Heyting, A. [1966] *Intuitionism: An Introduction*, North Holland Publishing Co., Amsterdam.

Turchin, V.F. [1977] *The Phenomenon of Science*, Columbia Univ.Press.

Turchin, V.F. [1980] *The Language Refal*, Courant Computer Science Report #20. New Yor University.

Turing, A.M. [1936] On computable numbers, with an application to Entscheidungsproblem. *Proc. London Math.Soc*. 2-42, 230-265.

Wiener, Norbert [1948] *Cybernetics*. John Wiley & Sons, New York.