

THE CYBERNETIC FOUNDATION OF MATHEMATICS
II. INTERPRETATION OF SET THEORY

Valentin F. Turchin
Department of Computer Science
The City College of New York

1. Basic Definitions

We start with a recapitulation of the basic definitions of our formalism which were introduced (sometimes semi-formally) in Part I of this paper. When we do not want to add anything to the previous definitions, we send the reader (implicitly) to Part I.

Our theory is using a formal language called *Refal* and a "machine on paper" called *the Refal machine*. Elementary syntax units of *Refal* are of two kinds: *special signs* and *object symbols* (or *just symbols*).

Special signs of *Refal* include :

- *structure brackets* '(' and ')';
- *activation brackets* '<' and '>';
- *free variables*, which are represented by a subscripted 's' (a *symbol variable*) or 'e' (an *expression variable*), e.g. s_1 , s_x , e_5 .

Object symbols used in *Refal* are supposed to belong to a finite alphabet, which is not, however, fixed once and forever. We shall use as object symbols:

- characters distinct from special signs,
- superscripted characters like F^i ,
- strings of characters underlined to form one (composite) symbol, e.g., \underline{fbc} .

We use capital italic letters *A*, *B*,... etc. as metasymbols to denote *Refal* objects and processes.

Refal's composite syntax units are as follows.

- An *expression* is an object which can be identified as one of:
(a) the empty string, which we may represent just by nothing, or

by the metasymbol [];

(b) a symbol (i.e. an *object* symbol, not a special sign);

(c) a variable;

(d) E^1E^2 , or (E^1) , or $\langle E^1 \rangle$, where E^1 and E^2 are expressions.

• A *term* is either a symbol, or a variable, or (E) , or $\langle E \rangle$, where E is an expression.

• A *pattern expression* is an expression which does not include activation brackets (but generally includes variables). A *process expression* is an expression which does not include variables (but generally includes activation brackets). An *object expression* is an expression which includes neither variables nor activation brackets. An L-expression is a pattern expression which:

(a) contains no more than one entry of every e-variable,

(b) contains no more than one e-variable on every level of bracket structure, i.e. *cannot* be represented as

$$E_1 e_1 E_2 e_j E_3 ,$$

where subscripted E 's are expressions. Examples of L-expressions:

$$Ae_1, BC(DE), e_1+(e_2)(e_3), s_1e_x s_1$$

Examples of pattern expressions which are not L-expressions:

$$e_1+e_2, (e_x)ABCe_x, e_1s_2((e_1-e_3))$$

• A *Refal sentence* is an object of the form $\langle L \rangle \rightarrow R$, where L is a pattern expression and R is an arbitrary (general) expression of Refal. The sign ' \rightarrow ' is just a symbol (not a special sign) which is used for visual convenience. L is referred to as *the left side*, and R as *the right side* of the sentence. The right side can include only such variables which appear also in the left side.

• A *list* of expressions E_1, E_2, \dots, E_n is the expression

$$(E_1)(E_2) \dots (E_n)$$

• A Refal program is a list of sentences.

The Refal machine has two information storages: the *program-field* and the *view-field*. The program-field contains a program, which is loaded into the machine before the run and does not change during the run. The view-field contains a process expression which changes in time as the machine works. The process expression in the view-field may be, in particular, an object expression, i.e. may not contain activation brackets. Then the Refal machine stops -- or, one might say, reproduces the same object expression indefinitely -- until a new run is initiated. Change, as we said above, comes only from activation brackets. This is our way of representing the abstraction of invariability, which lies at the root of the notion of an object. Our object expressions are linguistic representations of natural objects, which are supposed not to change with time. Concatenation and the use of structure brackets (parentheses) allow us to render the hierarchical structure of natural objects as they are built of certain elementary objects, which we represent by object symbols. To represent a change in time, i.e., a process, we enclose an object expression in activation brackets, and then the Refal machine will transform such expressions step by step, thus generating a linguistic process. If at some stage this process (i.e., the process expression in the view-field) becomes an object expression, we say that the process is *finite*.

Activation brackets may be nested; then they will be activated in a unique order using the principle 'inside-out, from left to right'. More formally, we define the *range* of an activation bracket as the subexpression limited by this bracket and the one paired with it. We define the *leading* activation bracket in a given expression as the leftmost sign < of those signs < which have no other signs < in their range. The Refal machine works by steps, each step being an application of one of the sentences from the program-field to the term in the view-field which starts with the leading activation sign; we call this term the *active term* of the process.

We say that an object expression E_o can be *syntactically recognized* as a pattern expression E_p if the variables in E_p can be replaced, observing the rules listed below, by object expressions called their values such that E_p becomes identical to E_o . The rules are as follows.

- (a) An s-variable s_l , where l is any index, can take as its value any symbol.
- (b) An e-variable e_l can take any expression as its value.
- (c) All entries of the same variable in E_p , i.e. variables with the same sign 's' or 'e' and the same index, must be replaced with the same value.

It can be shown that if E_p is an L-expression, then there is no more than one set of values for the variables in E_p such that their substitution transforms E_p into E_o , and there is an efficient algorithm which establishes whether E_o can be syntactically recognized as E_p , and in the case of a positive answer determines the values of the variables (see [Turchin 1980]).

Now we can describe the operation of the Refal machine. Each step starts with locating the active term in the view-field. If there is none, the Refal machine comes to a *normal stop*. Having found the active term, the Refal machine compares it with the consecutive sentences in the program-field starting with the first one in search of an *applicable* sentence. A sentence is applicable for an active term if the term can be (syntactically) recognized as the left side of the sentence. On finding the first applicable sentence the Refal machine copies its right side and replaces the variables there by the values they have taken in the process of recognition. The process expression thus formed is then substituted for the active term in the view-field. This ends the current step, and the machine proceeds to execute the next step. If there is no applicable sentence in the program, the Refal machine replaces the active term by the term $\langle ? \rangle$, which at each next step is replaced by itself again, thus generating an infinite process, which will be called *undefined*. This is a special process with the question mark symbolizing (in this context only) that if our linguistic process is intended as a

representation of a non-linguistic "real world" process then the former carries no information about the latter. It is important to note that an indefinite process is infinite.

The Refal machine may not be completely autonomous of its user. Some of the sentences in the view-field may have the form $\langle F^a \rangle \rightarrow R$, where F^a is a symbol, and R is an object expression which the user, and only the user, can change at any time. The expression $\langle F^a \rangle$ is the access function for the real-time process represented by the changing expression R . The most important real-time process is accessed as $\langle qns \rangle$. Its value Γ is the sum-total of the current mathematical knowledge of the user.

An expression without free variables defines a process: the one initiated by the Refal machine when this expression is put in its view-field. An expression which includes free variables (parameters) will be said to define a parametrized process. A parametrized process which never calls real-time access functions is a machine. A process initiated by a machine is *mechanical*. A process which calls real-time access functions is *metamechanical*. Thus when we are strict we use the concept of a machine in exactly the same way as it has been used since Turing: as an *autonomous* device. In this context, a parametrized process which is not known for certain not to call access functions, is not a machine. However, we may occasionally call a parametrized process a machine when we do not care whether it does or does not call access functions.

All processes we use have a symbol immediately after the opening activation bracket; it serves as a tag, or name, to distinguish different families of processes. We refer loosely to these tags and the processes so tagged as functions. Indeed, every parametrized process defines a partial function (which is recursive if the process is mechanical) whose value is the result of the process, i.e. the expression to be found in the view-field when (and if) the process comes to an end. A process $\langle FE \rangle$, where F is a tag and E is an expression, is denoted as $F(E)$ in a semi-formal functional notation, which we mostly use in the following.

We deal with two kinds of processes: searches and genera-

tors. The former have the structure $\langle FE \rangle$ at every stage; the latter $L\langle FE \rangle$, where L is a list of object expressions which are said to be produced by the generator. Searches and generators can be run in parallel, which is simulated on the "sequential" Refal machine as defined above. We use *metacode* to map the set of all Refal expressions on a set of object expressions. Speaking about Refal expressions we denote the metacode of E as $\uparrow E$. However, in the semi-formal notation we drop ' \uparrow '.

A prediction is the statement that a given search, say A , is finite. We represent it as $A!$ According to our fundamental principle, a proposition is meaningful if it can be interpreted as a hierarchical generator of predictions. One such proposition is the statement that the search A is infinite; we denote it as $A?$. To represent in this way all logical connectives and quantifiers, we found it necessary to introduce cognitive processes $\gamma(P)$ and $\bar{\gamma}(P)$, the former seeking to prove P as true according to our current knowledge Γ (which may change itself), and the latter looking for contradiction of P with Γ .

Predictions are directly verifiable. This holds, however, for those predictions only which state the finiteness of a deterministic mechanical process. By introducing metamethodical processes we have undermined the simple and safe notion of a prediction. If A is metamethodical, the statement $A!$ has no immediate meaning, because A is not a deterministic process: it depends on the user's will. This was the note on which Part 1 ended.

To finish up with preliminaries, a table follows which translates the usual logical notation into the corresponding propositions of our theory. The translation of a logical proposition P will be denoted as $[P]$. Primitive predicates are translated according to their meaning. For composite propositions, the translation rules are:

$[\neg P]$	=	$\bar{\gamma}([P])!$
$[P \& Q]$	=	$\underline{\text{and}}([P], [Q])$
$[P \vee Q]$	=	$\underline{\text{or}}(\gamma([P]), \gamma([Q]))!$

$[P \rightarrow Q]$	=	$\underline{\text{if}} \forall ([P]) \underline{\text{! then}} [Q]$
$[(\forall x)P(x)]$	=	$\underline{\text{all}}(x: [P(x)])$
$[(\exists x)P(x)]$	=	$\underline{\text{sch}}(x: \forall ([P(x)]))!$

2. Objective Interpretability

There are two reasons why predictions about metamechanical processes cannot have the same direct meaning as predictions about mechanical processes. First, a metamechanical process is not deterministic. It is not defined ~~beforehand~~ for all the in advance future, but only for the past, up to the present moment. Its further development depends on decisions to be taken by the user. Second, different users may have taken different decisions in the past, so even the past of a metamechanical process is not quite definite.

Then is there anything definite about metamechanical processes? Or does the freedom of the user's will render every statement about such processes meaningless?

It would do so, if we put no constraints on the user's choices and decisions, allowing him everything. But the value of r provided by the user is not completely arbitrary, even though it is not uniquely defined. By the consistency rule discussed in Part I, if we find that some P is not true we cannot say: "so what, let us still add it to the knowledge". By the completeness rule, if we figured out that some P is true, we cannot say: "to hell with it, we still do not want to add P to the knowledge, ever". We have to add it. Because of these restrictions we can separate some objective facts about metamechanical processes, which do not depend on the user's will as long as he sticks to the rules, from those features which do depend on his will and hence have no objective interpretation.

The only objective property of the process $\forall(P)$ in case it is finite is that it is finite; an analogous statement can be made about $\bar{\forall}(P)$. All the rest may vary from user to user, and from one moment in real time to another. Take any property of a process that characterizes its definite stage, e.g. that the

number of terms in the process expression after 25 steps is even. It is easy to write a formal proposition (it will be a prediction) which expresses this property; let it be P . If the process is mechanical, P has a definite (objective) value; it is either true or false. If the process calls gn, then depending on the current value of r the number in question may be different for different users; even for the same user, it may be even today, odd tomorrow, then again even, and so on. The proposition P is not *objectively interpretable*, it depends on the very process of cognition, not solely on the processes which are being cognized.

Less obvious cases of objectively uninterpretable propositions are $\gamma(P)?$, and $\bar{\gamma}(P)?$, where P is an interpretable proposition referring only to mechanical processes. Compare $\gamma(P)!$ and $\gamma(P)?$. The former is a prediction which does not tell us anything about any specific stage of the cognitive process, but only the fact that the process comes to a successful end, which characterizes P as true. The latter is a generator of propositions about *the stages of the cognitive process*; it does not qualify as objectively interpretable.

We define interpretability inductively. The base of induction involves only those processes that do not call cognitive processes and are deterministic. If A is such a search then $A!$ and $A?$ are interpretable. If G is such a generator, and none of the propositions it produces refers to cognitive processes, then G is interpretable.

As discussed above, if P is in the base (purely mechanical), then the predictions $\gamma(P)!$ and $\bar{\gamma}(P)!$ are objectively interpretable: the former says that P is true, the latter that it is false, while the truth-falseness of P has a quite definite objective meaning. Generalizing this reasoning, consider a process A , which at some stage initiates a cognitive process $\gamma(P)$ or $\bar{\gamma}(P)$. If the results of A depend only on the fact that the cognitive process with P as the argument ultimately stops, then such a process A can be interpreted in objective terms, specifically, the results will be conditional on the truth (the case of γ) or falsehood (the case of $\bar{\gamma}$) of the proposition P . Generalizing

further, we can understand by P any proposition whose interpretability has already been proved. Thus we come to the concept which will be referred to as *strong* interpretability, to distinguish it from a version which will be introduced later as *weak* interpretability.

Definition of **strong interpretability**

- I.1 If A is a deterministic model-time process with no access to real-time processes, then $A!$ and $A?$ are interpretable (atomic) propositions.
- I.2 If A is such a process that whenever it initiates a cognitive process of the form $\forall(P)$ or $\exists(P)$,
- (1) P is an interpretable proposition, and
 - (2) the results of A , i.e.
 - in case when A is a search, the fact that it is finite, (and if it is finite its final stage) and
 - in case when A is a generator, the set it generates,do not depend on any stage of the cognitive process but merely on the fact that it is finite or infinite,
- then the process
- A
- is interpretable.
- I.3 If A is an interpretable search, then $A!$ is an interpretable proposition.
- I.4 If G is an interpretable generator which produces only interpretable propositions, then G is an interpretable proposition.
- I.5 A proposition is interpretable only if it can be proved interpretable by definitions (I.1) to (I.4).

The concept of interpretability can be compared with the concept of invariance in physics. When we write equations of theoretical physics, we use some reference system, thus it becomes ingrained in the meaning of the equations. Yet the most important physical quantities are those which are invariant with regard to coordinate transformations. We ascribe to them more

objectivity, because they do not depend on our choice of the system of reference. Thus we choose a reference system and use it to create models of reality, but then look for those features of these models which are independent of the reference system. This is the only way to give a precise meaning to the concept of objectivity: not to ignore the fact that our knowledge always has a subjective component, but to construct invariants which are independent of at least some part of our arbitrary choices.

Cognitive functions are sort of reference systems of mathematical knowledge. Our analysis has shown that they are present in the meaning of mathematical propositions, like reference systems are present in the meaning of the equations of physics. Conversely, reference systems of physics can be called cognitive functions, or devices.

We shall prove now the following theorem, where we abbreviate "strong objective interpretability" to "interpretability":
Theorem 1. For every interpretable proposition we can construct an interpretable process which will lead to labeling the proposition as either true or false. We call this process *objective evaluation*; it formalizes our intuitive notion that a proposition is false if it produces at least one false prediction, and is true otherwise.

Proof. We take as the starting point the law of excluded middle for the special case of a prediction: every mechanical search is either finite or infinite, i.e. for all A ,

$$(EM_1) \quad \underline{or}(A, \neg(A?))!$$

Our intuition accepts this principle without hesitation, so we can simply take it as an axiom.⁶⁾ It also can be put in a relation to the basic rules defining cognitive process by the following proof. It is impossible to refute (EM_1) , because to do that we must prove that both A and $\neg(A?)$ are infinite. But by proving that A is infinite, we make $\neg(A?)$ finite. Since (EM_1) may never lead to a contradiction, we add it to our knowledge by the completeness rule, and it becomes formally true.

We now prove Theorem 1 by induction, using the definition of interpretability.

(a)Base. Consider an atomic proposition $A!$, or $A?$, where A is a mechanical search. The process of objective evaluation is this: Run the searches A and $\neg(A?)$ in parallel and label the proposition T or F depending on which branch ends. Because of (EM_1) , this process is always finite. It labels the atomic proposition T if it is intuitively true, and 'F' if it is intuitively false.

(b)Induction on generation. Consider a proposition-generator P which calls no cognitive functions. We can assume that as a process it is infinite. If P is finite, we modify it so that instead of stopping it goes on infinitely without producing new members. All the propositions produced by P are interpretable and, by the inductive hypothesis, have a definite objective evaluation. Intuitively, P is true if it produces only true propositions. We can construct a process which tests this. Every time that a proposition is produced by P we apply the process of objective evaluation to it. By the induction hypothesis, it is always finite. If the result is F , we stop. If it is T , we go on running P . This defines a certain process; let us denote it as A . It is not mechanical, but we can define a process A' by "cutting out" all the calls of the evaluation process and leaving only their final stages, symbols T and F . Process A' is mechanical. Because of objectivity of the evaluation process, A' is the same for all users. By (EM_1) it is either finite, in which case we label P as F , or infinite, and then P is labeled T .


(c)Induction on cognitive function calls. Consider an interpretable search A . In running this search, whenever $\nu(P)$ or $\bar{\nu}(P)$ is called, the proposition P is interpretable and, by the induction hypothesis, has an objective evaluation. Modify A as follows. When $\nu(P)$ is met, initiate the process of objective evaluation of P . It is always finite. If the result is T , replace $\nu(P)$ by any object expression (according to point 1.2 in the definition of interpretability, the further development of A will not depend on it). If the result is F , replace it by any infinite process. Let the process modified in this way be A' . Reasoning as in case (b),

we can apply (EM_1) to A' . If A' is finite, we label A' as true, otherwise as false.


Consider an interpretable proposition-generator G . Replace $\forall/\bar{\forall}$ -calls as above. The resulting generator can be treated as in point (b). Thus we again are able to construct a process, the objective interpretation of G , which is finite, produces T or F, and corresponds to our intuitive understanding of objective truth-values of propositions. This completes the proof. \square

Let A and B be processes. A is *immediately semantically dependent* on B , if one of the stages of A includes $\forall(B!)$, $\forall(B?)$, $\bar{\forall}(B!)$, or $\bar{\forall}(B?)$ as an active subexpression. For A to be interpretable, all processes on which A semantically depends must be interpretable, and their interpretability must have been established *prior* to the consideration of the interpretability of A . We define the relation of *semantic dependence* as the transitive closure of semantic dependence. If a process semantically depends on itself we refer to this situation as a *semantic recursion*. For a parametrized process, semantic recursion, like the usual recursion, may be finite or infinite. A process which generates infinite semantic recursion is uninterpretable.


For a better insight into the structure of propositions, we shall use their *semantic graphs*. The following is the definition of the elements of semantic graphs.




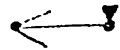
A proposition-generator which starts producing something. A semantic graph is a directed graph with nodes (dots) representing propositions and arcs (lines) representing processes. Unless the direction of an arc is indicated explicitly, it is from left to right and from top down. If a line peters out, it means that the graph does not show what will happen later.

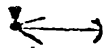


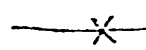
A proposition-generator which branches into three parallel processes.

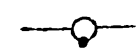
- 


A proposition-prediction that a search is finite; and the beginning of that search.
- 

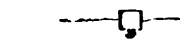
A search branches into two parallel searches.
- 

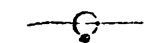
A branch of a generator produces a prediction.
- 

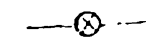
A branch of a search ends.
- 


A process becomes infinite. The part to the right of the cross never materializes.
- 


A process calls $\nu(P)$ and goes on. P is represented by a dot, i.e. as a proposition-generator.
- 

A process calls $\nu(A!)$ and goes on. We do not distinguish between a proposition-generator which generates one prediction, and that prediction.
- 

A process calls $\bar{\nu}(P)!$ and goes on.
- 

A ν -call is known to be finite. Analogously for $\bar{\nu}$.
- 

A ν -call is known to be infinite. Analogously for $\bar{\nu}$.
- 

A process has more branches than shown in the graph.
- 

An infinite loop in the process.

Examples of semantic graphs. Fig.1 is the graph for $A!$ and $B!$, where A is finite and B infinite. Fig.2: the proposition if $\nu(B?)!$ then or $(\bar{\nu}(A!), \bar{\nu}(A?))!$ with A finite and B infinite.

Fig.3 represents proposition isr! where process isr is defined by:

$$\underline{isr} \rightarrow v(\underline{isr?})$$

('isr' stands for 'infinite semantic recursion').

There are a number of simple transformations that can be done over a semantic graph without changing it in a significant manner. An infinite recursion loop can be replaced by a cross; two consecutive stages of a process can be merged into one; an infinite branch of a generator can be eliminated.

Consider the walks in the semantic graph of a proposition P , which start at node P . When such a walk passes over from a call $v(Q)$ or $\bar{v}(Q)$ to Q we have a *metasystem transition*. The number of metasystem transitions on a walk is its *semantic length*. A walk is *semantically finite* if its semantic length is finite, and *semantically infinite* otherwise. For a proposition to be strongly interpretable it is necessary and sufficient that its semantic graph contains no semantically infinite walks.

The requirement of strong interpretability can be weakened. If a generator produces at least one interpretable and false proposition, it can be labeled as false even though some of its other branches are uninterpretable processes. In contrast, for a generator to be true, all of its branches must be interpretable and true. Also, a search can be labeled as finite if at least one of its branches is interpretable and finite, even though other branches may be uninterpretable. A search is interpretable and infinite only if all its branches are interpretable and infinite. We can make this extension of the concept of interpretability because we examine parallel branches of processes in parallel. When examining an uninterpretable branch, the evaluation function will work infinitely in a futile attempt to get to the bottom of semantic recursion; meanwhile, another branch may lead to a definite result.

We shall call this extended interpretability *weak*. The process of determining that a proposition is weakly interpretable is unseparable from labeling it as true or false, i.e. its objective

interpretation. We define this process below as labeling the nodes of the semantic graph of the proposition according to 10 labeling rules.

Labeling rules



LR1. A call $\forall(P)$ with P labeled T is marked finite.



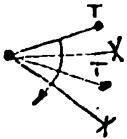
LR2. A call $\forall(P)$ with P labeled F is marked infinite.



LR3. A call $\forall(P)$ with P labeled T is marked infinite.



LR4. A call $\forall(P)$ with P labeled F is marked finite.



LR5. If every branch starting from a proposition-generator either leads to a proposition labeled T, or is infinite, this proposition-generator is labeled T.



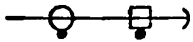
LR6. If at least one branch starting from a proposition-generator leads to a proposition labeled F, then this proposition-generator is labeled F.



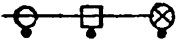
LR7. If at least one search starting from a prediction node is finite, then this prediction is labeled T.



LR8. If every branch starting from a prediction node is infinite, then this prediction is labeled F.



LR9. A branch is finite if it ends, and all $\forall/\bar{\forall}$ -calls on it are marked finite.



LR10. A branch is infinite if one of these cases takes place:

(a) there is an infinite $\forall/\bar{\forall}$ -call such that all $\forall/\bar{\forall}$ -calls before it are finite;

(b) there is an infinite recursion loop with all the $\forall/\bar{\forall}$ -calls on the branch marked finite



For an example of labeling see Fig.2. The prediction in Fig.3 is left unlabeled: it is uninterpretable.

Rules LR6 and LR7 do not require that all branches are labeled; some branches may call uninterpretable processes. Thus a proposition-generator or a prediction may be interpretable even though the processes it involves are only *partially interpretable*. A proposition which generates at least one false proposition is false no matter how we interpret -- or fail to interpret -- all other propositions generated by it. A search is finite if we know that at least one branch has led to a stop, no matter what happens to all the other branches. Note an essential difference between true and false propositions when we allow weak interpretability: a false proposition-generator can produce uninterpretable propositions, while a true one cannot. We shall see

later that weakly interpretable processes play an important role in set theory.

As in the case of strong interpretability, our intuitive notion of weak interpretability defined by labeling rules can be translated into a formal process of objective evaluation. Because of different requirements to interpretability in the cases of a true and a false generators, as well as in the cases of finite and infinite parallel searches, we have to run two parallel processes for weak interpretability where for strong interpretability we had one. For a generator, one process is the running of the parallel branches in an attempt to prove that the proposition produced on one of the branches is false; the other process is the proving that proposition produced on every branch is true. If the first process ends, the generator-proposition is interpretable and false. If the second process ends, it is interpretable and true. If neither of the two ends, the proposition is uninterpretable. The labeling of searches as finite or infinite is treated analogously. Thus Theorem 1 holds for weak interpretability as well; a weakly interpretable proposition is either true, or false. In the following, when speaking of interpretability without the strong-weak attribute, we shall mean weak interpretability, as the more general case.

The equivalence of P being true and $\forall(P)$ being finite is part of the definition of $\forall(P)$. The falseness of P , as defined by the labeling process is not the formally same as the finiteness of $\forall(P)$. However, their equivalence is established by the following theorem, the proof of which is straightforward:

Theorem 2. An interpretable P is false iff $\forall(P)!$.

If a proposition is not interpretable, the process of labeling does not ever end, so such propositions are neither true nor false. When we start labeling, we do not know in advance whether the proposition will turn out interpretable or not. Until the process stops, we never know whether we still did not reach the end, or the process is really infinite. This situation is familiar from the theory of algorithms (recursive functions). There is no mechanical process with a guaranteed end which could tell for any

mechanical process whether it is finite or infinite. However, the whole point of introducing cognitive functions ν and $\bar{\nu}$ is to provide a process (metamechanical though) which solves the problem unsolvable for mechanical processes. Indeed, $\nu(A?)$ stops if and only if the mechanical process A is infinite. Of course, a solution given by a metamechanical process is not a real solution, because at any moment in real time you can only run mechanical approximations to ν , not the full ν itself. Cognitive functions provide a formalism to exactly *define* problems; in order to solve them, we still have to construct mechanical processes, algorithms.

Reasoning by analogy with the theory of algorithms, one would wish to construct a formal definition of objectively interpretable proposition using cognitive functions to recognize infiniteness of processes when necessary. Indeed, we defined the concept of interpretability in words. Let us translate it into our formalism, i.e. define a process int(P) (no doubt, metamechanical) which terminates if and only if P is interpretable.

It may come as a surprise, but this is impossible. The concept of objective interpretability cannot be objectively defined -- in its full volume. To understand why it is so, recall the way we defined objective interpretability, and try to write the corresponding Refal recognizer.

Let us take, for simplicity, strong interpretability. It is easy to define a function, call it sem, which recognizes the fact that a given process P semantically depends on a process Q . i.e. one of the stages of P activates one of: $\nu(Q?)$, ... etc. So, P is semantically dependent on Q if and only if sem(P,Q)!. By our verbal definition, P is strongly interpretable, if Q is strongly interpretable whenever sem(P,Q)!. We formalize this condition by universally quantifying over all possible Q . The proposition which expresses this, is:

all($Q \in \text{exp}$: if sem(P,Q)!then int(Q)!)

where exp is the generator of all Refal expressions. Now, the

process $\text{int}(P)$ only has to check that this proposition is true. We come to the following definition:

$$(1) \quad \text{int}(P) \rightarrow \neg(\text{all}(Q \in \text{exp}: \text{if sem}(P,Q) \text{ then int}(Q)!))$$

But it does not satisfy the requirement of being interpretable for all P . Inside the \neg -call we have $\text{int}(Q)$ with Q which runs over all expressions, P , in particular. For every P which semantically depends on itself, we immediately get uninterpretability. An obstacle of this kind will prevent us from formalizing the universal concept of interpretability in any thinkable way. This is testified by the following theorem:

Theorem 3. There exists no parametrized process $I(P)$ such that $I(P)$ is objectively interpretable for every P , and is finite iff P is objectively interpretable.

Proof. Suppose that there is such a process $I(P)$. Define a process A by:

$$A \rightarrow \text{if } I(A!) \text{ then } \bar{\nu}(A!)$$

Since $I(A!)$ is interpretable, it is either finite, or infinite. Suppose it is finite. Then $A!$ must be interpretable. But if $I(A!)$ is finite, A calls $\bar{\nu}(A!)$, so it is uninterpretable. Suppose $I(P)$ is infinite. Then $A!$ must be uninterpretable. But in this case A never calls anything but $I(A!)$, which is interpretable. So, A must be interpretable. The inescapability of a contradiction proves the theorem. \square

But if we cannot formalize the concept of interpretability, is not our verbal definition meaningless, or contradictory?

The reason for the non-existence of $I(P)$ is that we defined interpretability inductively; in fact, we defined a hierarchy of interpretability concepts, with the propositions never referring to cognitive functions as the ground level. This does not make the word 'interpretability' unusable; one should simply keep in mind that it refers to a hierarchy of definable concepts, not one concept. (This, of course, reminds of [Tarski 1933], even though

int^3, \dots etc. All these concepts (parametrized processes) are inter

our approaches are very different philosophically.)

We can define 0-interpretability int^0 using the process of checking that cognitive functions are never called. Then we define 1-interpretability int^1 by modifying (1) so that in the left side we have the new concept, while the right side uses only the concepts already defined:

$$\text{int}^1(P) \rightarrow \forall(\text{all}(Q \in \text{exp}: \text{if sem}(P, Q) \text{ then } \text{int}^0(Q)))$$

In this way we can define an infinite series of concepts: int^2, \dots interpretable.

This is not the end, of course. If a process is n -interpretable, the semantic length of the branches of its semantic graph never exceeds n . We can define the concept of ω -interpretability, which allows the branches on the semantic graph to be of arbitrary finite length, then $\omega+1$ -interpretability using ω -interpretability in the right side of (1), etc. This is, essentially, how the ordinals of set theory emerge in our theory. The next theorem explains why the set of all sets is non-existent:

Theorem 4. There exists no objectively interpretable generator which produces all objectively interpretable expressions, and only them.

Proof. If there were such a generator G , we could immediately construct $I(P)$, whose existence is denied by Theorem 3. One has only to run G and stop if and when it produces P . \square

Objectively interpretable propositions constitute our universe of discourse. Every expression inside this universe is meaningful. No one outside is. If there were an expression U which defined this universe by generation or recognition, then the now achievable part of this universe, i.e. the set of expressions known to be interpretable, could be simply obtained by replacing Γ in U by Γ_i . Our current universe of discourse would be a definite function of our current knowledge. But this is not the case. We can expand the available domain of discourse by creating (discovering, if you wish) new spheres in the never-to-be-fully-covered universe, even without expanding the current

knowledge Γ . This is a real-time process which, like the knowledge process, cannot be expressed by a fixed object. To represent it in our formalism, we introduce one more access function: lqs (for 'logos'). Its value, for which we shall use the metavariable Λ , is, at every moment in real time, an interpretable generator which produces all the expression proven interpretable by this moment.

So, we have two real-time processes in mathematics. The sum-total of our knowledge, the 'gnosis', is accessed by gns. The sum-total of the expressions known to be meaningful, our universe of discourse, the 'logos', is accessed by lqs. They are interrelated: Γ is always a subset of Λ ; on the other hand, our ability to prove interpretability depends on our current knowledge. But neither of them is a simple derivative of the other. We have two degrees of freedom here, not one.

Adding a new access function lqs, we have expanded the class of metamathematical processes; they now can call lqs, not only gns. To distinguish those processes which may call only gns, we shall call them γ -metamathematical.

Objective interpretation is based on our intuition of the separability of the object and the subject of knowledge. When we deal with quantum-mechanical phenomena this intuition deceives us. The object and the subject of knowledge are not completely separable in the quantum-mechanical measurement. Our functions γ and $\bar{\gamma}$ can be seen as measurement procedures, of a kind. We took pains to separate the results of these 'measurements', i.e. truth values of propositions, from our state of knowledge. Our theory allows 'interpretable' propositions only; this leads to the usual two-valued logic. It is possible that a more general theory can be built, which would not limit itself to those propositions we call interpretable, thus overstepping the boundaries of traditional logic. This possibility occurred to the author under the influence of the ideas of the wave logic developed by Yuri Orlov [1978, 1982]. Orlov's ideas can probably be used in trying to expand the present theory aiming at description of subatomic phenomena.

3. Logical Paradoxes

The famous paradox of the liar was known to the ancient Greeks in the following form (see [Beth 1968]): Epimenides, the Cretan, says: "I am lying." Now, if what he says is true, then he is lying, and what he says is false. If what he says is false, then he is not lying, which means that what he says is true. Thus what he says can neither be false, nor true. A more modern version is: "The proposition expressed by this very sentence is false."

Let us formalize the liar paradox. We have here a proposition which states its own falseness and nothing more. Since this proposition refers to itself, we must give some name to it, otherwise reference will be impossible. Let the name be P. The relation between a name and its meaning is defined in Refal by a sentence of the form:

$$\langle P \rangle \rightarrow \dots$$

where the dots in the right side stand for the meaning. The meaning of a proposition is a generator of predictions. The proposition P produces only one prediction: that P is false, i.e. $\bar{\bar{v}}(*P)!$. Hence $\langle P \rangle$ is this generator:

$$(1) \quad \langle P \rangle \rightarrow (\bar{\bar{v}}(*P)!) , \quad \text{semi-formally:} \quad P \rightarrow \bar{\bar{v}}(P)!$$

Looking at this definition, one can see that it is the most straightforward and clear formalization of the proposition "P says that P is false." It is, of course, uninterpretable, so it has no objective meaning, and one should not be surprised that there is no way to assign a definite truth-value to it.

The semantic graph of the proposition P is presented in Fig.4. We see an infinite semantic recursion here; it is impossible to label this graph starting from the bottom. If we still want to make some labeling, we notice that if one of the nodes

representing P is labeled T , then the nodes preceding and following it must be labeled F , which is inconsistent. Still, it is possible to label the whole infinite graph. There are two ways to do it, though; they are shown in the Figure. If we use one way, the root of the graph is labeled T , if we use the other, it is labeled F .

Suppose we change $\bar{\nu}$ to ν in the definition of P :

$$(2) \quad \langle P \rangle \rightarrow (\nu(*P))! , \quad \text{or:} \quad P \rightarrow \nu(P)!$$

According to our definition of interpretability, P is still uninterpretable. But (2), unlike (1), does not lead to contradiction. Yet it is quite justified that we declare P uninterpretable, because we still cannot assign to P a unique truth-value. Its semantic graph is presented in Fig.5. If any of the nodes is labeled T , then all of them must be labeled T ; if any is labeled F , all must be labeled F . Thus, it is possible to label the graph consistently, but, again, there are two ways to do it, and the root can be labeled both T and F . This may be called *the paradox of the saint*. The saint says that what he is saying is true, and that is all he says. He gives no evidence in support of this statement. We have only his word for that. If what he is saying is true, then what he is saying is true. If what he is saying is false, then what he is saying is false. There is no contradiction in either case, and there are no reasons to consider this proposition either true or false. It is uninterpretable.

There is one more paradoxical definition which is pertinent to the concept of interpretability. The propositions P and $\nu(P)!$ are logically equivalent, meaning that if one is true, the other is true too, and if one is false, the other is false. Replace $*(\nu*(P))!$ in (2) by $*(P)$:

$$(3) \quad \langle P \rangle \rightarrow (*(P))$$

This generator is, of course interpretable, because it has no reference to cognitive processes. What is its meaning and truth-

value? The proposition $\langle P \rangle$ produces exactly one proposition, which is itself. This proposition is not a prediction, but a generator of predictions. To find what is the ultimate set of predictions produced by $\langle P \rangle$ we must run $\langle P \rangle$ again. But this returns us to the starting point, so P never produces a single prediction. This proposition is empty. According to our definition of labeling, it must be labeled T .

Although P and $\neg(P)!$ are logically equivalent, they are not semantically equivalent. It is not the same to state P and to state that P is true. The former is, generally, a generator of predictions; in our case it turns out that it produces nothing. The latter is a prediction. It cannot be empty, it certainly tells us something. In this case, it tells us that P produces no false predictions.

The translation of the definition (3) into our natural language is " P says what it says". Our intuition readily accepts the judgement we passed above: this proposition is true, but empty.

Consider one more paradox, formulated first by Grelling (we use [Beth 1968] as the source). Let us call an adjective *autological*, if and only if it can be validly applied to itself; *heterological* if it cannot. The adjectives "English" and "polysyllable" are, according to these definitions, autological; the adjectives "French" and "red" are heterological. Now, consider the adjective "heterological". What type is it? If it is autological, then, by definition, it is applicable to itself, i.e. heterological. If it is heterological, it cannot be applicable to itself; then it is not heterological, and, therefore, autological.

This paradox takes us to the world of natural languages, so we have to represent this world in our theory. Like Refal objects, the objects of natural languages are strings of characters, with which some processes are associated; these processes determine the semantics of the language. Unlike the case of Refal, semantical processes of natural languages are part of the functioning of the human brain, not a mechanical device. In terms of the construction of the Refal machine, these processes are

external; yet nothing prevents us from using in our theory predictions and generators of predictions referring to these processes.

Convenient representation of external objects and processes by Refal objects is a matter of agreement. Let us represent processes and objects by composite Refal symbols formed by underlining corresponding adjectives and nouns. Since the objects of natural languages may be both non-linguistic and linguistic, we should be able to easily distinguish between these categories. Quotes are used for this purpose in natural languages; they mark linguistic objects. Let, therefore, apple be the representation of the apple, while 'apple' be that of the word 'apple'.

The simplest and a reasonably complete test to determine whether somebody understands the language is putting to this person questions which require answers 'yes' or 'no', and seeing whether the answers are correct. This is the basis for the widely used formalization of the semantics of natural languages in predicate calculus. In particular, the semantics of adjectives of the English and other European languages is represented in this approach by one-place predicates (properties).

In our theory we deal with semi-predicates, not predicates. This is, as we discussed in Part I, a more general case; the decision process either comes to a halt, which is interpreted as the positive answer, or fails to halt, which means that the property is not there. For instance, the adjective 'red' can be represented in Refal by the external process $\langle \underline{\text{red}} e_x \rangle$. According to the usual meaning of this notion, the process is this: I look at the object represented linguistically by e_x and if and when I come to the conclusion that e_x qualifies as red, I stop the process; if I cannot come to this conclusion, the decision process goes on infinitely. (Instead, I could define being red as reflecting the light of certain wavelength. The semantic process then will include the necessary measurements with a spectrometer. That will be a different meaning.)

The adjective 'English', in contrast to the adjectives like 'red', is applied to linguistic, not external, objects. We can define it by listing all the linguistic objects that come from

the English language:

<English 'red'> → T
<English 'brown'> → T
<English 'apple'> → T
... etc.

One of the sentences will be

<English 'English'> → T

Analogous definitions can be given to the adjectives 'French' etc.

Now it is a straightforward matter to define the adjectives 'autological' and 'heterological', which we abbreviate to A and H, respectively:

<A 's_p'> → <s_p's_p'>
<H 's_p'> → <¬ *(s_p's_p')!>

Note an important difference between the two concepts. Being autological can be defined without recourse to cognitive functions, while being heterological cannot. As a result, the first definition is obviously interpretable (in the assumption that all other adjectives are interpretable, of course), while the second requires an analysis of interpretability. From the definition of A, <A 'red'> immediately becomes <red 'red'>, which cannot halt because the adjective 'red' is applicable only to external objects, not to linguistic objects. Applying A to the adjective 'English', we have the search <English 'English'>, which, according to the above definition, halts in one step. 'English' is an autological adjective. 'French' is not; 'Francais' is autological.

Let us see now whether the adjective 'autological' is autological. Starting <A 'A'>, we enter an infinite loop. Thus 'autological' is not autological. No contradictions or paradoxes are met here. If, however, we try to determine whether 'heterologi-

cal' is heterological -- and this is what the Grelling paradox is about -- the result will be different. Running $\langle H 'H' \rangle$ produces a cognitive function call with the metacode of $\langle H 'H' \rangle$ as the argument; thus the process becomes uninterpretable. In this case again, the paradox is caused by a meaningless definition, which is not allowed in our theory.

The two truth-values enter the syntax of mathematical logic symmetrically. Negation can be viewed as a sort of symmetry transformer; this view is quite useful in a Boolean algebra. Thinking solely in terms of the conventional mathematical logic, it is hard to understand why of two parallel definitions, which differ only by an added negation, one can be safely used, while the other blows up the whole theory. Our semantic approach gives the explanation. There is no symmetry between affirmation and negation. Negation always involves one more metasystem transition: the one from simply stating or using a process, to exploring it for contradictions. This asymmetry is embedded in the syntax of our theory, which allows us to ban the definition of 'heterological' without banning (unnecessarily) the definition of 'autological'.

4. Formal systems and theories

The formal systems we are going to consider will be constructed in the framework of a metasystem common to all of them. This metasystem is the Refal machine, together with a number of functions (machines) defined in its program field. All logical machines defined above are in that number, plus a few more which will be defined later.

A formal system is defined if:

- (1) a Refal representation for a number of parametrized processes is defined; and
- (2) a proposition is given which is believed to be true, and is referred to as the *knowledge* of the formal system.

Some of the parametrized processes of the formal system may be defined by a group of sentences in the program field, i.e. as

Refal functions. Others may be left undefined, or defined partially. Even if not defined, a process can be an object of study and knowledge. We may not be able to reproduce all the stages of a process, but still know that it is finite or infinite, or that if it is finite then a certain proposition must be true, etc.

The knowledge of a formal system F_i contains in a condensed form all the propositions that can be proven true in F_i . We shall denote the knowledge of a specific formal system F_i by Γ_i , and the corresponding cognitive functions by ν_i and $\bar{\nu}_i$. Thus a proposition P is provably true in F_i if and only if $\nu_i(P)$ is finite. P is provably false if and only if $\bar{\nu}_i(P)$ is finite.

Our concept of a formal system differs in two ways from the usual concept. First, we do not distinguish between axioms and inference rules: they are united in the concept of a generator. The knowledge Γ_i of our formal system is analogous to the axioms of a usual formal system, but because of the nature of our propositions, no additional rules of inference are necessary. When a proposition P is among those hierarchically produced by Γ_i , it corresponds to the derivability of P from Γ_i in a usual formal system. When P added to Γ_i produces a contradiction, it corresponds to the derivability of $\sim P$, the negation of P .

Second, our concept of a formal system is, starting from the basic definitions, *semantical*, in contrast to the usual purely *syntactical* concept. Therefore, in addition to the usual concepts of consistency and completeness applied to formal systems, we also apply the concept of correctness: a formal system is correct if it hierarchically produces only true predictions.

A formal system is, essentially, a machine which encapsulates only a certain amount of knowledge. You cannot expect more output from a generator, than you have put into it through its definition. Goedel's result that no formal system can produce all the true statements about a machine which is sophisticated enough is intuitively taken as natural with our concept of a formal system, while it comes as a surprise with the usual concept.

We shall distinguish between a formal system and a *theory*. While a formal system can be fully represented by an object (the

metacode of the machine), a theory is a real-time process resulting from human effort to gain new knowledge. Formal systems we create are stages of theories. Some theories may be completed by creating a formal system which gives answers to all possible questions meaningful in the theory. But this is rather an exception. The most important theories are infinite real-time processes.

Among the objects and processes of a theory we distinguish primary objects and processes: those which we treat as a given reality and wish to explore. Other objects and processes are created as exploration tools. The primary objects of a theory may be defined either by listing them when their number is finite, or by defining a machine which generates all of them. Primary processes may be defined either directly and completely by Refal sentences, in which case we call the theory cybernetic, or indirectly by propositions believed to be true and called axioms, in which case the theory is axiomatic. Hybrides of these two kinds of theories are also possible.

We can illustrate the difference between cybernetic and axiomatic theories by taking arithmetic as example.

In cybernetic arithmetic (known also as *recursive arithmetic*) the numbers are strings:

$\emptyset, \emptyset 1, \emptyset 11, \emptyset 111, \dots$ etc. ,

or their equivalents. Operations on numbers are machines: the adding machine, the multiplying machine, and possibly others. All these machines are *defined*. The adding machine, for instance, is defined by the sentences:

$$\begin{aligned} \langle +(e_x)(\emptyset) \rangle &\rightarrow e_x \\ \langle +(e_x)(e_y 1) \rangle &\rightarrow \langle +(e_x)(e_y) \rangle 1 \end{aligned}$$

When we add numbers we run this machine or one of its more sophisticated equivalents, like a pocket calculator.

In axiomatic arithmetic there is one number constant \emptyset and

an *undefined* function $\langle \underline{s} e_x \rangle$ which produces the 'next' number after e_x . Repeated application of the function \underline{s} produces all possible numbers. The functions of addition and multiplication are also undefined, but they comply with a number of axioms. The axioms relating functions \underline{s} and $+$ are:

$$x + 0 = x$$

$$x + \underline{s}(y) = \underline{s}(x + y)$$

They resemble the sentences defining addition in cybernetic arithmetic, but conceptually they are different: they are a part of the knowledge, not the machinery, of the theory. The function of equality which is used in the axioms is not defined either; all we know about it is its well-known properties stated as axioms.

The strength of classical logic as compared to intuitionist logic comes from the more permissive treatment of the cognitive functions. Intuitionistic logic considers the user's knowledge r to be a definite expression r_i , at least for the time of discourse (what we called a *static* interpretation of human knowledge). In this interpretation, the law of excluded middle in its general form:

$$(EM_2) \quad \underline{or}(\underline{v}(P), \underline{\bar{v}}(P))! \text{ for every proposition } P$$

is, certainly, not true. Indeed, \underline{v} in (EM_2) is understood by intuitionism as \underline{v}_i . But by Goedel's theorem, for every formal system there is a proposition G which is neither provable, nor refutable in this formal system. Thus, both $\underline{v}(G)?$, and $\underline{\bar{v}}(G)?$ take place, which contradicts to (EM_2) with $P = G$.

Classical logic bases its proofs on the concept of a growing r (dynamic interpretation); it allows the index i in r_i to go into infinity. We shall show, first, that with this interpretation Goedel's theorem does not falsify (EM_2) .

Goedel's theorem establishes two facts. Firstly, for every formal system r_i there exists such a proposition G_i that

(a) $\underline{\text{or}}(\gamma_i(G_i), \bar{\gamma}_i(G_i))?$

Secondly, G_i is true, so that we can add it to the knowledge and get a new correct formal system Γ_{i+1} , in which, of course, G_i is provable:

(b) $\underline{\text{or}}(\forall_{i+1}(G_i), \bar{\forall}_{i+1}(G_i))!$

If we simply "take the limit" of (a) and (b) for $i \rightarrow \infty$, we get two contradictory propositions. This is a situation familiar from the calculus, when the correct answer depends on the order in which two interrelated variables are treated in the jump to the limit. In intuitionist logic we first fix the index i of the formal system and let (EM_2) to be produced with all possible P . Then it will generate at least one proposition, namely G_i , such that (EM_2) is false. In classical logic we use γ and $\bar{\gamma}$ to denote the limit of γ_i and $\bar{\gamma}_i$ as $i \rightarrow \infty$. The law of excluded middle is a proposition-generator which produces (EM_2) for every proposition P . Thus P comes first, and then we interpret (EM_2) by seeing γ and $\bar{\gamma}$ as the corresponding limits. Then for every Goedel proposition G_i , there is a stage of the real-time cognitive process at which it is proven, thus (EM) is not contradicted.

Theorem 5. With the dynamic interpretation of cognitive functions, the law of excluded middle (EM_2) is true for every interpretable P .

Proof. By Theorem 1, every interpretable proposition P is either true or false in objective evaluation. If it is true, $\gamma(P)$ is finite; if it is false, $\bar{\gamma}(P)$ is finite by Theorem 2. Thus, (EM_2) is always finite. \square

When we create a formal system we take a proposition Γ_i as its knowledge and define the access function $\langle \text{qns} \rangle$ as

$\langle \text{qns} \rangle \rightarrow \Gamma_i$

Now the function $\nu(P)$ called by proposition-generators, which was undefined before, becomes a completely defined recursive function which we denote as $\nu_i(P)$; and $\bar{\nu}(P)$ becomes $\bar{\nu}_i(P)$. Note that the replacement of $\nu/\bar{\nu}$ by $\nu_i/\bar{\nu}_i$ takes place only for the purpose of generation. The ultimate product of proposition-generators, the predictions, can still include $\nu/\bar{\nu}$ -calls; there is no need to replace them. (The replacement would signify a change in interpretation from the dynamic to a static one).

We should now explore the relation between the 'precise' function $\nu(P)$ and its 'approximation' $\nu_i(P)$. What we want, of course, is that the formal system be correct, i.e. $\nu_i(P)$ be finite only when $\nu(P)$ is finite (P objectively true). This relationship is established in the following theorem, which is crucial for the whole theory we are developing.

Theorem 6 (Correctness theorem). If Γ_i is true then $\nu_i(P)$ for any interpretable P is finite only if P is true, i.e. a formal system with the knowledge Γ_i is correct.

Proof. Let $\nu_i(P)$ be finite and suppose that P is false. The process $\nu_i(P)$ is the running of the generator Γ_i until it produces P . Consider the branch B_i of Γ_i which has produced P (to be referred to as the *derivation branch* for P), and compare it with the corresponding branch B in the semantic graph of Γ_i . They are different only in that every call $\nu(Q)$ in B is replaced by $\nu_i(Q)$ in $B_i(P)$, and every $\bar{\nu}(Q)$ is replaced by $\bar{\nu}_i(Q)$. The branch B_i has no more than a finite number of $\nu_i/\bar{\nu}_i$ calls. Let them be:

$$(*) \quad \nu_i(Q_1), \dots, \nu_i(Q_n), \bar{\nu}_i(Q'_1), \dots, \bar{\nu}_i(Q'_m)$$

We can take every Q_r and find a derivation branch in Γ_i which produces Q_r . And we can take every Q'_s and find two derivation branches in $(\Gamma_i \text{ and } Q_s)$ which produce a contradictory pair of atomic propositions, $A!$ and $A?$. Since each of these branches is finite, we can again construct derivation branches for the $\nu_i/\bar{\nu}_i$ calls they involve (if any). Since the process of producing P from Γ_i is finite, we shall ultimately come to a finite deriva-

tion tree for P (see Fig. 6).

Consider the v_i/\bar{v}_i calls (*). The corresponding v/\bar{v} calls in the semantic graph of Γ_i cannot all be finite because it would mean that Γ_i produces a false proposition P and is, therefore, false. Hence either there is a false Q_r for which $v_i(Q_r)$ is finite, or there is a true Q'_s for which $\bar{v}_i(Q'_s)$ is finite, (or both). In the first case we again face a situation where a true Γ_i produces a false proposition, this time it is Q_r . In the second case a true proposition (Γ_i and Q'_s) produces a pair $A!$, $A?$ of atomic propositions from which one is false: the same situation again. In both cases the new derivation tree is a subgraph of the original tree. Since it is finite, this situation cannot repeat unlimitedly. Sooner or later we must come to a true proposition which generates a false proposition. This contradiction proves the theorem.

Corollary. If Γ_i is true, the formal system which takes Γ_i as its knowledge is consistent.

There are two possible situations with regard to what can be treated in the theory as an *object*. (1) The objects of the theory may be predefined at the outset of that theory. If their number is finite, they can be simply listed. If their number is potentially infinite, a machine can be defined which generates all of them. This situation is traditionally known as a first-order theory. The objects of such a theory are completely separated from the propositions about the objects. (2) The set of the objects of the theory may not be predefined, and the propositions of the theory may, in their turn, become new objects. This situation is referred as a higher order theory. Set theory is a theory of infinite order. Indeed, the concept of a set is essentially identical to the concept of a proposition: when we define a set we define a predicate of being an element of this set, and vice versa. In set theory we define sets, which then become legitimate new objects. This conversion of a proposition into an object can be repeated indefinitely.

In Part I we interpreted the language of classical first-order logic. To prove the correctness, and therefore, consis-

tency of classical logic, we have to prove that its axioms and inference rules are true propositions. We also should consider how Goedel's theorem is formulated and proved in our theory. Limitations of space do not allow us to do it in this paper. We would like, however, to give the reader an idea of how our semantic approach differs from the usual syntactic ("formal") approach by discussing one of the most fundamental logical principles of our theory.

It is the verification principle, which states that the proposition

(Ver) all($x \in \text{ser}$: if $x!$ then $x!$)

is true and must be included into the knowledge r_i of every theory T_i . Here ser is the generator of all searches which can be expressed in T_i . Using a metavariable, (Ver) can be formulated as stating that

(Ver') if $A!$ then $A!$

is true for any search A from T_i .

If we try to translate this principle into the language of formal logic, we come to the trivial axiom

$\text{finite}(A) \rightarrow \text{finite}(A)$

which is of little, if any, use. In our system, however, the verification principle is far from being trivial. Let us see what is the full scope of propositions it hierarchically produces. We add to our knowledge r_i the proposition (Ver) which produces the propositions (Ver')

with all possible searches $A!$ expressible in the theory. When r_i starts working, every proposition (Ver') starts working, and if the search A in it is defined and finite, it produces the prediction $A!$. If the search is undefined or infinite, it produces nothing. Thus all those and only those predictions $A!$ will be

produced by Γ_i , for which Λ is defined and finite. The verification principle formalizes the fact that the finiteness of a search which is defined mechanically can be directly verified, at least in principle. (Ver) produces all those propositions which can be proved true through verification.

5. Set theory. Extensionality and Regularity

We proceed now to the interpretation of set theory. We identify the concept of a set with the concept of a generator. Since we allow the use of real-time cognitive processes γ and $\bar{\gamma}$, we limit set generators to interpretable processes, otherwise we shall not be able to interpret the membership of an object in a set.

As we know, there is no generator, mechanical or γ -metamechanical, which could produce all interpretable generators. There is no set of all sets. The generator Λ of all legitimate objects of set theory, its universe of discourse, is an independent real-time process; we shall use for the access function of Λ the same symbol lqs as in our general theory.

When an object and a set are given, we must be able to establish whether the object is an element of the set. It is easy to define a function, let it be called elm, such that the search elm($E \in G$) stops if and only if the expression E is among the expressions generated by G .

However, this straightforward concept of being an element is not the one adopted in set theory. It is applicable only when the expression E represents one of the *primary objects*, or *ur-elements* of the theory, by which we mean those objects (if any) which are not sets, so that their 'physical' identity as expressions is the necessary and sufficient condition of being identical as objects of theory. But most important objects of set theory are, of course, sets. Set theory uses the extensionality principle to define the identity of sets. According to this principle, two sets are declared identical, or equal, if and only

if every element of one set is also an element of the other. Consequently, the identity, or equality of sets is not the same as the identity of the Refal expressions which represent them. Indeed, it is easy to define in Refal two different processes which will generate the same objects.

To comply with the extensionality principle, we must distinguish between ur-elements and sets, and use the concept of set equality when deciding whether a given set is among the objects produced by a given generator.

The set of all ur-elements may be different in different versions of set theory (it may be, in particular, empty). The only requirement on this set is that we should be able to distinguish an ur-element from a set. We define an ur-element as any Refal expression which includes no asterisks *. This immediately makes ur-elements distinguishable from set representations because the latter have the form $*(E)$.

In case of infinite sets, their equality, unlike the physical identity of the expressions which represent them, cannot be directly established. A reference to some proof, i.e. to a knowledge, once again becomes an implicit part of semantics.

Equality of sets is defined through a double inclusion:

$$(1) \quad (S=T) \equiv ([S \text{ sub } T] \& [T \text{ sub } S])$$

The relation of inclusion (being a subset) is defined by

$$(2) \quad (S \text{ sub } T) \equiv \text{all}(x \in S: \text{el}(x \in T)!)$$

It is a universally quantified proposition, and so is $S=T$. Now, let $\text{el}(X \in S)$ be the process of establishing that X is an element of S in accordance with the extensionality principle. To establish that X is an element of S , we have to compare X with elements of S , and determine whether there is one $Y \in S$ which is equal to X . So, when running $\text{el}(X \in S)$, we call in parallel $\gamma(X=Y)$ for all elements Y of S . Thus $\text{el}(X \in S)$ is semantically dependent on all the propositions $X=Y$ with a Y from S . Using

symbol \gg to denote semantic dependence, we can represent this by the formula:

$$(3) \quad \underline{el}(X \in S) \gg X=Y, Y \in S$$

(The usual notation $X \in Y$ stands for $\underline{el}(X \in Y)$!).

From (1) and (2) we derive:

$$(4a) \quad X=Y \rightarrow \underline{el}(Z \in Y), Z \in X$$

$$(4b) \quad X=Y \rightarrow \underline{el}(Z \in X), Z \in Y$$

Combining (3) with (4a) and (4b) we have two semantic dependencies:

$$(5a) \quad \underline{el}(X \in S) \gg \underline{el}(Z \in Y), Y \in S, Z \in X$$

$$(5b) \quad \underline{el}(X \in S) \gg \underline{el}(Z \in X), Y \in S, Z \in Y$$

A process which is semantically dependent on itself (infinite semantic recursion) is uninterpretable. From (5a) we see that we are immediately in trouble if S is among the elements of itself: $S \in S$. Indeed, putting $X=Y=Z=S$, we find that $\underline{el}(S \in S)$ is uninterpretable. From (5b), our process becomes uninterpretable if $X=Z$ and $S=X$, therefore $X=Z=S$. Since Y is an element of S and $Z=S$ is an element of Y , such a situation will arise if $S \in Y$ and $Y \in S$.

We call an el-sequence a sequence of sets Y_1, Y_2, \dots etc., such that for every $i > 1$, $Y_{i+1} \in Y_i$. A set is *regular* if it is interpretable, and an el-sequence of sets which starts with S can only be finite.

Theorem 7 (Regularity Theorem). The process $\underline{el}(x \in S)$, where x is an ur-element or a regular set and S is a regular set, is interpretable.

Proof. According to the definition of the function el, the only source of possible non-interpretability is the semantic recursion in function el itself. Consider a pair (X, S) which is the argument of an el call. Denote by Z' any element of the set Z .

According to (5), the semantic recursion in function el can be schematically presented by two formulas:

(6a) $(X,S) \gg (X',S')$

(6b) $(X,S) \gg (S'',X)$

If X and S are regular sets or ur-elements, then any possible sequence of the calls of function el can only be finite. Therefore all of them have a definite objective interpretation which can be established starting from the bottom.

Theorem 8. If a set is regular, all its elements are regular. Conversely, if all elements of an interpretable generator S are regular, S is also regular.

Proof. Indeed, should an element T of a regular set S be not regular, an infinite el-sequence starting with T would exist. Then we have only to add S to it to prove that S is not regular either. The second part is proved by noticing that should we have an infinite el-sequence for S , we could delete S and get an infinite el-sequence for one of its elements.

So, for a set to be regular, it is sufficient and necessary that all its elements are regular. Therefore, all regular sets can be constructed inductively starting with sets which include only ur-elements.

It should be stressed that regularity becomes necessary only because function el is defined according to the extensionality principle. The concept of a set which has itself as one of its elements is not contradictory in itself. For instance, this generator:

(7) $\langle \underline{\text{self}} \rangle = (*(\underline{\text{self}}))$

is interpretable as a process. It generates exactly one element which happens to be the metacode of this very process. If we based the concept of being an element of a set on the literal identity of expressions, as in function elm, it would be true that

self elm self

But we would not be able to use function el with such sets. The necessity of regularity arises from extensionality.

Now we limit the objects of our theory to ur-elements and regular sets only, which guarantees the interpretability of the el processes. It follows immediately from the definition of regularity that there is no (regular) set generator that could produce all regular sets. We access the generator of all legitimate objects of set theory, i.e. ur-elements and sets, through the function lqs. At every moment in real time the process $\langle \text{lqs} \rangle$ yields a specific interpretable generator Λ_i , which produces all those sets that are *already known to be legitimate*, i.e. interpretable and regular.

A set is regular if at some stage of the development of theory it becomes producible by lqs. This is the completeness rule with respect to Λ . Since the expression lqs is not among those produced by lqs, it does not represent a regular set. This may seem paradoxical, because at every moment in real time the expression Λ_i yielded by lqs is a regular set generator. It even may seem inconsistent. Indeed, is not Λ , which is represented by lqs, the union of all Λ_i ? A union of regular sets is regular, hence Λ is regular.

This argument, however, is not valid, because it manipulates -- in the spirit of the Platonist-formalist marriage -- with fictitious entities and operations. According to the intuitive set theory, sets exist as some "ideal" entities; operations on them are, correspondingly, also ideal. This is not a sound ground for a mathematician. So, formalism declares these entities and operations "abstract" (as if one can exercise abstraction on non-existent entities), so that now only their properties, which are expressed as axioms, matter. In this framework, the above argument leads, indeed, to a contradiction. To avoid it, the concept corresponding to our Λ is either banned altogether, or is declared a class, not set. Even if this eliminates the contradic-

tion, it explains nothing.

Our theory deals with real (linguistic) objects and processes. Symbol lqs does not stand for any ideal entity. It stands for itself. Its meaning is in how it is manipulated by the user. Sets are linguistic processes. To define a set means to actualize such a process. To define the union of the infinite series $\Lambda_1, \Lambda_2, \dots$ etc., we must have a generator of this series. But we know that there is no such generator.

This explains why the non-regularity of Λ is consistent. Let us now see why it is true. The role of the symbol lqs is to serve a sort of hook on which to hang all the generators proven to be regular. Consider a certain moment in real time. Let Λ_i be hanging on the hook. This means we have proven that every object produced by Λ_i is a regular set (or an ur-element). We do not yet know whether Λ_i is regular (otherwise it would have hanged on the hook already, and then Λ_i would produce Λ_i , which is impossible). Suddenly we realize that we can apply Theorem 8. We do it, and prove that Λ_i is regular. The very moment we have proven it, we hang Λ_i on the hook, by the definition of the access function lqs (it should be the last phase of the proof). After we did it, Λ_i is proven to be regular, but it is something else that is hanging now on the hook, namely $\Lambda_{i+1} = \Lambda_i \cup \{\Lambda_i\}$. Thus at no time have we proven that the current Λ is regular. The universal set generator Λ is a consistent notion in our theory because we consider the process of doing mathematics in its dynamics, as it should be. Being Λ is always one step before being regular.

6. Basic set constructors. Paradoxes

The language of set theory is, in its essence, a programming language. Like other programming languages, such as FORTRAN or REFAL, the set-theoretical language gives us the means to create linguistic processes (set generators in the case of set theory) which we use to model natural phenomena. Unlike computer programming languages, the language of set theory includes the means to communicate with the real-time processes Λ and Γ .

The role of basic operations of computer languages is played in set theory by *set constructors*. These are machines defined in the Refal metasystem and used to create new set generators. A set constructor must be such that when its arguments satisfy certain stated requirements, the generating process is interpretable and the set produced -- regular. In the following we define and discuss the basic set constructors necessary to arrive at the present-time set theory.

The general procedure for using an expression E as a set-theoretical object is as follows.

1. See if E is generated by lgs. If it is, use it.
2. If E is not produced by lgs prove that E represents an interpretable generator and that whenever it produces a set S , every element T of S either was in lgs from the beginning, or has already been produced by E at an earlier stage in model time.
3. If you succeed, add a generator which produces E to the defining list of lgs. You can use it now.

We need, first of all, the means to create arbitrary finite sets out of objects which are already in existence. Since a finite set can be represented simply by the list of its elements, we can define a trivial function fs (for 'finite set') which takes a list as its argument and produces its members one-by-one:

$$\begin{aligned} \langle \underline{fs}(e_1)e_2 \rangle &\rightarrow (e_1)\langle \underline{fs} e_2 \rangle \\ \langle \underline{fs} \rangle &\rightarrow \end{aligned}$$

Examples. A set generator for a set of two elements A and B is $\langle \underline{fs}(A)(B) \rangle$, hence what is (A,B) in the usual set-theoretical notation will be $*(\underline{fs}(A)(B))$ in the strict notation of our theory, and $\underline{fs}(A,B)$ in the semi-formal notation. The empty set is $*(\underline{fs})$. The set $(A,(A,B))$ is $*(\underline{fs}(A)(\forall(\underline{fs}(A)(B))))$ in our theory. Each step of set formation brings one metacode transformation more.

It is easy to define function uni which implements set-

theoretical union. You simply run two or more set generators in parallel.

Set theory requires the existence of at least one infinite set, namely, the set which contains the empty set \emptyset as its element, and together with any element x contains also the element formed as the union $x \cup \{x\}$. Thus the elements of this set are:

(1) $\emptyset, \langle \emptyset \rangle, \langle \emptyset, \langle \emptyset \rangle \rangle, \langle \emptyset, \langle \emptyset \rangle \rangle, \langle \emptyset, \langle \emptyset \rangle \rangle, \dots$ etc.

To construct a generator producing (1), we define:

$\text{inf}(X) \rightarrow (X) \text{inf}(\text{uni}(X, \text{fs}(X)))$

In every step, function $\text{inf}(X)$ produces X and calls itself with $X \cup \{X\}$ as the next argument.

It is easy to see that with any interpretable argument e_x the process $\langle \text{inf } e_x \rangle$ is interpretable, and if e_x is a regular set, then it produces only regular sets. Hence $\text{inf}(\text{fs}())$ is the desired infinite set. This is a constructor without parameters which gives us exactly one set.

Our next constructor will produce sets with elements selected for a certain property. Its format is:

$\text{set}(x \in S; H(x)!)$

which is read: the set of all those elements x of the set S for which the process (search) H depending on x as a parameter is finite. The search $H(x)$ will mostly be the proving of a certain property P of x , or its negation, i.e. $\nu(P)$ or $\bar{\nu}(P)$. The set machine works as follows. G is run step by step. Each time that it produces an object x , this object is substituted into the search H and the search is run in parallel with the continued running of G . Those branches for which $H(x)$ stops produce the corresponding object x .

The construct

$$(2) \quad T = \underline{\text{set}}(x \in S; H(x)!)$$

is a regular set if and only if the following three conditions are satisfied: (a) the generator S is interpretable, (b) the search $H(x)$ is interpretable for every element x produced by S , and (c) all elements x of S for which $H(x)$ is finite represent ur-elements or regular sets.

If S is regular, then the necessary and sufficient condition for (2) to represent a legitimate set is that the process $H(x)$ is interpretable for every possible element of S .

Can we use lqs in the role of S in the set constructor? Consider

$$T = \underline{\text{set}}(x \in \underline{\text{lqs}}; H(x)!)$$

Although lqs is not regular, it is interpretable, because it produces only interpretable generators. So, condition (a) is satisfied. Consider condition (b). Take the case when the search $H(x)$ is $\neg(P(x))$ or $\bar{\neg}(P(x))$. As mentioned before, this is the most typical use of the set constructor. In particular, the set used by Russell to come to his famous paradox, namely

$$R = \underline{\text{set}}(x \in \underline{\text{lqs}}; \bar{\neg}(P(x))!)$$

with $P(x) = x \in x$, is of that type. For R to be interpretable, the property $P(x)$ must be interpretable for every $x \in \underline{\text{lqs}}$, and since $P(x)$ is within a $\bar{\neg}$ -call, this interpretability must be proven before and independently of the interpretability of R . That is, R is semantically dependent on x and $P(x)$, for every $x \in \underline{\text{lqs}}$.

At first glance it may seem that we could prove the legitimacy of R by the following reasoning. lqs produces only regular sets; therefore $P(x)$ and $\bar{\neg}(P(x))$ are interpretable for every x . Then R is interpretable and regular. This reasoning, however, is faulty. The error is that a metamechanical process is thought of as if it were mechanical. When we say "lqs produces only regular

sets", the implication is that lqs is somehow given to us as an external, completely definite reality. But it is not. It is part of our (the user's) activity. The correct reading is: "we should manipulate our machinery in such a fashion that lqs always produces only regular sets". The correct reading of "is R regular?" is: "can we include R in Λ without violating the rules?"

To answer the last question, we tentatively add R to lqs and see whether this will violate the rules. It will. When we add R to lqs, R becomes semantically dependent on itself. It is uninterpretable. The set constructor cannot be used with the universal generator lqs. We can collectivize objects by an arbitrary property only if they belong to a definite regular set.

Without coming into detail at the present time, we contend that all paradoxes of set theory are resolved in our theory in the same way we resolved Russell's paradox: by showing that they use uninterpretable propositions. The failure to interpret set theory in a constructive way has been the result of thinking and arguing about metamathematical processes as if they were mechanical.

When set theory is defined axiomatically, the axioms are chosen in order to avoid paradoxes. This is hardly a satisfactory way to found a theory. We start our theory from a certain conception of what the meaning of mathematical propositions is. We do not have to do anything to avoid paradoxes. As far as we use only meaningful propositions the paradoxes simply do not appear.

We saw that the set constructor with the universal generator lqs cannot be used to collectivize objects by an arbitrary property. However, if we specify the collectivizing property in a certain way, namely by putting:

$$P(x) = x \text{ sub } S$$

where S is a definite regular set, then we still can form a universal set. This set, i.e. the set of all subsets of S , known as the *powerset* of S , plays a most important role in Cantor's set theory. It deserves a special constructor:

$$\text{pow}(S) \rightarrow \text{set}(x \in \text{lqs}: \neg(x \text{ sub } S)!))$$

If S is a regular set, $\text{pow}(S)$ is also a regular set. To prove it, add $\text{pow}(S)$ to Λ . The process $\text{pow}(S)$ is *weakly interpretable*. Its semantic graph is presented in Fig.7. It includes a semantically infinite path, but it does not prevent us from labeling all the propositions involved. x_1, x_2 , and other elements of $\text{pow}(S)$ may or may not be elements of S , but S itself certainly is not an element of S , being regular. Thus

$\text{pow}(S) \text{ sub } S$

is interpretable and false; $\text{pow}(S)$ is not produced by $\text{pow}(S)$, while all other x 's produced by it are regular because they have been in lqs before the introduction of $\text{pow}(S)$. This proves the regularity of $\text{pow}(S)$ for any regular S .

By allowing pow into our theory, we add a new type of set generators. The meaning of the concept of objective interpretability remains the same, but the user has now two 'degrees of freedom': gns and lqs .

7. Functions

In set theory, a *function* is a certain subset of the Cartesian product of certain sets. As everything in set theory, this definition is a static representation of an intuitive concept that is inherently dynamic, procedural. Intuitively, a function $f(x,y)$ is a device which for every given x initiates a process (search) which ultimately halts, yielding the corresponding y .

Our theory returns to the concept of function its intuitive procedural content. An objectively definable function is a parametrized search which for every set of parameters (arguments) is objectively interpretable, and such that if it is finite on more than one parallel branch, then the final passive stage (the value of the function) is the same on all branches. It can be easily

shown that this definition is equivalent to the usual set-theoretical definition when expressed in terms of our theory.

We identify function with the procedure of its "computation"; the quotes are here because we extend the concept of computation by allowing reference to real-time processes gns and lgs. Such a procedure may not be executable in a computer; it is a definition, not an algorithm. According to the current terminology, a function is computable if it can be defined by a purely mechanical process, an algorithm. We should better call such functions mechanical. Non-computable functions should be properly called non-mechanical.

To compute a function $f(x)$ defined through a metamechanical process we have to replace the calls of Γ and Λ by the best of our 'gnosis' and 'logos' for today: Γ_i and Λ_i . The resulting mechanical function $f_i(x)$ will be referred to as an *implementation* of the function $f(x)$. Where the exact function calls $\gamma(P)$ or $\bar{\gamma}(Q)$, its implementation calls $\gamma_i(P)$ and $\bar{\gamma}_i(Q)$. the following theorem, the proof of which is left to the reader, is for functions what Correctness theorem is for propositions:

Theorem 9. Let $f(x)$ be an objectively definable function, and $f_i(x)$ its implementation. For a given x , if the search $f(x)$ is infinite, then $f_i(x)$ is also infinite; if $f(x)$ is finite, then $f_i(x)$ may be either finite or infinite, but if it is finite then its value $f_i(x)$ is equal to the value of $f(x)$.

Function f_i may be defined on just a part of the domain of f . If f is non-mechanical, it will always be a part, not the whole domain. However, for those x for which $f_i(x)$ is defined, its value will be exactly the same as that of $f(x)$. No implementation of an objectively defined but non-mechanical function is complete; at any moment in time there will be an argument x for which the search $f_i(x)$ is infinite. But for any x from the domain of f there exists such an implementation f_i of f that $f_i(x)$ is finite and produces the correct value.

One can see that our definition of a function is, basically, constructive: a function is a computational procedure which may employ metamechanical processes. This extension of the concept of

computation is fully justified. Metamechanical processes are distinguished from mechanical not in that they appeal to some "transcending" powers or use non-existent omniscient creatures ("oracles"), but in that they include the activities of the subject of knowledge, the user of the mechanical device. Turing, who introduced machines on paper into mathematics, identified computation with the activity of an *autonomous* machine. But computational processes which we see around are autonomous only during some finite stretches of time: say, while a computer program is working. But it is the user of the computer who has written the program, and who will soon throw it away and write a better one. It is the user who proves theorems and bases new algorithms on them. Mathematics is being done by human beings, not computers. A metatheory of mathematics should start with an acknowledgement of the existence of metamechanical processes. Computation is a metamechanical process. Mechanical, algorithmic computation is only a special case.

Calling non-mechanical functions "non-computable" is confusing; it is something of a contradiction in terms. Non-computability implies that the function cannot be computed. Then is it a function? We know, however, that every objectively defined function can be computed, even though at the present time we may not know *how*.

There is a fundamental difference between the use of parallel processes in propositions and in functions. The interpretation of a parallel search in a proposition depends only on the finiteness or infiniteness of the branches, but not on their final stages. We could agree that the final stages of all searches the finiteness of which is asserted in propositions are always identical to the symbol T; or we could simply ignore them. Therefore, a parallel search is interpretable if every branch is interpretable. A functional parallel search is interpretable only if all finite branches produce one and the same result.

A functional search which runs an interpretable generator is, generally, uninterpretable, because it may cut the process at

some moment, and both this moment, and the result of the search may depend on "the competition" of the branches, which is implementation dependent. In particular, the concept "the first member produced by the set generator" is uninterpretable if the generator involves metamechanical processes. So is, of course, "the second member", etc. While the belonging to the set defined by a metamechanical generator is objectively interpretable, the order in which the members are produced is not. This explains why sets of set theory must be unordered if we want to consider not only recursively-enumerable sets.

In logic, a functional dependence is a predicate $F(x,y)$ which has the property:

$$(*) \quad (Ax)(Ey)(Az)[F(x,z) \equiv y=z]$$

It states that for every x there is exactly one y such that $F(x,y)$ holds.

Given an interpretable functional dependence $F(x,y)$, we can build the corresponding computational process using the Refal function fun defined as follows:

$$(**) \quad \underline{\text{fun}}(x: F(x,y)) \rightarrow \underline{\text{sch}}(y \in \underline{\text{igs}}: \vee(F(x,y)))$$

With a given x , function fun tries every element y of igs, i.e. every legitimate object of the theory known up to date, looking for such a y that $F(x,y)$ is true. The expression y is then given out as the value of the function (see function sch in Part I).

The search defining $f(x: F(x,y))$ is objectively interpretable because igs produces only interpretable expressions, and $F(x,y)$ is supposed to be interpretable for every interpretable x and y . We have no problems of the kind we had with the set function, because **(**)** defines a search, not a new object (it is no constructor). Since $F(x,y)$ has the property **(*)**, there will be no more than one branch that is finite. Therefore, fun is an objectively defined function.

We shall need a Refal generator which computes a function

and outputs its value as its single element. This generator is: $\underline{gfs}(\underline{fun}(x: F(x,y)))$, where \underline{gfs} ("generator form search) is defined trivially:

$$\langle \underline{gfs} e_x \rangle \rightarrow (e_x)$$

5. The ZF axioms

Using the set constructors we defined above and adding a few more we can explain the meaning of the axioms of ZF and show why they are true. A more formal proof of consistency of the ZF system will be published later.

There are no ur-elements in the ZF system. All objects are sets.

I. Extensionality axiom. Sets having the same elements are equal:

$$(EXT) \quad (Ax)[x \in a \equiv x \in b] \rightarrow a = b$$

This is one part of our definition of equality between sets. Using the reversed implication one can easily prove that the equality so defined is, as required, reflexive, symmetric, and transitive.

II. Axiom of the empty set. There is a set which has no elements:

$$(EMP) \quad (Ea)(Ax)[\neg(x \in a)]$$

This set is $\underline{fs}()$.

III. Separation axiom. For every set a and every property $P(x)$ of sets there exists a set whose elements are those and only those elements of a which have the property P :

(SEP) $(\exists b)(\forall x)[x \in b \equiv x \in a \ \& \ P(x)]$

This set is: $b = \underline{\text{set}}(x \in a : \forall(P(x)))$

IV. **Pairing axiom.** Given any sets a and b , there exists a set c whose elements are exactly a and b :

(PAIR) $(\exists c)(\forall x)[x \in c \equiv (x=a \vee x=b)]$

This set is: $c = \underline{\text{fs}}(a, b)$

V. **Sum-set axiom.** For every set a there exists a set b , whose elements are exactly those objects occurring in at least one element of a :

(SUM) $(\exists b)(\forall x)[x \in b \equiv (\exists y)[y \in a \ \& \ x \in y]]$

We introduce a new constructor to satisfy this axiom: $\underline{\text{sum}}(S)$. It runs S , takes every element of S as a set generator, and runs them all in parallel. $\underline{\text{sum}}(S)$ is obviously regular if S is regular.

VI. **Powerset axiom,** for every set a there exists a set b the elements of which are exactly the subsets of a :

(POW) $(\exists b)(\forall x)[x \in b \equiv x \underline{\text{in}} a]$

The set b is $\underline{\text{pow}}(a)$.

VII. **Axiom of infinity.** There exists a set which includes the empty set and with every set x includes $x \cup \{x\}$:

(INF) $(\exists a)[\emptyset \in a \ \& \ (x \in a \rightarrow (x \cup \{x\}) \in a)]$

The set a is $\underline{\text{inf}}(\underline{\text{fs}}())$.

VIII. Axiom of replacement. The image of a set under an operation (functional dependence) is again a set. More precisely, if a is a set and $F(x,y)$ is a formula such that for every x from a there is exactly one y such that $F(x,y)$, then there exists a set the elements of which are exactly those y 's for which an $x \in a$ exists such that $F(x,y)$:

$$(REP) \quad (Ax)(Ey)(Az)[F(x,z) \equiv y=z] \rightarrow \\ (Eb)(Ay)[y \in b \equiv (Ex)[x \in a \ \& \ F(x,y)]]$$

To expect that a set required by (REP) exists, we must first prove that the antecedent of the implication is true, i.e. for every x there is a corresponding y . Suppose we did. Then we also have proved that all the y 's are regular, which is to have proved that all of them are generated by the current Λ_i "hooked" on lgs. In the definition of the function $\underline{fun}(x: F(x,y))$ we can replace lgs by Λ_i ; denote the resulting function as $\underline{fun}_i(x: F(x,y))$; it is equivalent to $\underline{fun}(x: F(x,y))$. Now we can construct the set required by (REP) as follows. Run set a , and for every element x generate the corresponding y using $\underline{qfs}(\underline{fun}_i(x: F(x,y)))$. It is, of course, regular.

IX. Axiom of regularity (or foundation). Every non-empty set is disjoint from at least one of its elements:

$$(REG) \quad a \neq \emptyset \rightarrow (Eb)[b \in a \ \& \ (Ax)[x \in a \rightarrow \neg(x \in b)]]$$

If every element of a has another element of a as its element, then there is an infinite (cyclic or acyclic) sequence of sets such that each next set is an element of the preceding one, which starts with a . Since a is regular this is impossible.

X. Axiom of choice. If a is a set the elements of which are non-empty sets, then there exists a function f with domain a such that for every member b of a it is true that $f(b) \in b$.

Such a function is referred to as a *choice function*; let us denote it as cho. We could try to construct cho as a machine which runs b as a generator and stops the moment it produces the first element. This element becomes cho(b). Since no element of α is an empty set, this function is defined on the whole set α .

Function cho, however, cannot be legitimately used in set theory. As we saw above, a function which employs running a generator is, generally, uninterpretable. The interpretability of cho(b) can be guaranteed only when b is countable; for this case, however, the axiom of choice has little significance because it can be proved as a theorem: one only needs to map the set b on natural numbers and pick up the element which corresponds to number 1. If b is uncountable it calls lcs, which changes in real time. Let the element of b picked up by the function cho at a certain moment be b_1 . We cannot guarantee that later in real time cho will pick up b_1 again. The belonging to b is objectively interpretable, but the order in which the elements of b are generated is not.

The following theorem we put forward tentatively, because not everything about the interaction of Λ - and Γ -processes is yet clear to us.

Theorem 10. There exists no objectively definable choice function.

Proof. Let us denote by $\Lambda_1, \Lambda_2, \dots$ the sequence of universes of discourse as they occur in real time as a result of using current values of Λ and Γ . Suppose that an objectively definable function cho does exist. Let cho₁, cho₂, ... etc. be its implementations under $\Lambda_1, \Lambda_2, \dots$ etc. By Theorem 9, if the search cho_i(b) is finite, then cho_j(b) for all $j > i$ are also finite and have the same value as cho_i(b), i.e. the true value of cho(b). This must be true for any sequence of Λ_i , provided that $\Lambda_i \text{ sub } \Lambda_{i+1}$. We are going to prove Theorem 10 by constructing such a sequence $\Lambda_1, \Lambda_2,$

Let α in the axiom of choice be the set of all non-empty subsets of some set S . Take a set $b \in \alpha$. Take some set Λ_2 and two of its non-empty subsets Λ_1 and Λ'_1 , such that $\Lambda_1 \cap \Lambda'_1 = \emptyset$, $b \cap \Lambda_1 \neq \emptyset$, $b \cap \Lambda'_1 \neq \emptyset$. (If b is uncountable, this is always

possible). We can now consider two Λ -sequences: one starts with $\Lambda_1, \Lambda_2 \dots$; the other with $\Lambda'_1, \Lambda_2 \dots$. Consider implementation $\underline{\text{cho}}_1(b)$ with the first sequence. Function $\underline{\text{cho}}_1(b)$ can depend on b only through the function $\underline{e}_1(x \in b)$. Hence $\underline{\text{cho}}_1(b)$ depends on b only through $\underline{e}_1(x \in b)$. Since those and only those elements are available to \underline{e}_1 , which are in Λ_1 , $\underline{e}_1(x \in b) = \underline{e}_1(x \in b \cap \Lambda_1)$. Therefore, $\underline{\text{cho}}_1(b) = \underline{\text{cho}}(b \cap \Lambda_1)$. Since $b \cap \Lambda_1$ is a subset of S and is not empty, function $\underline{\text{cho}}$ must be defined on it; let its value be c_1 . Reasoning in the same way for the second Λ -sequence, i.e. $\Lambda'_1, \Lambda_2, \dots$ etc., we find that the implementation for the first stage Λ'_1 is: $\underline{\text{cho}}'_1(b) = \underline{\text{cho}}(b \cap \Lambda'_1) = c'_1$. Since Λ_1 and Λ'_1 are disjoint, $c_1 \neq c'_1$. Let $\underline{\text{cho}}_2(b)$ for the second stage of both Λ -sequences be c_2 ; it is distinct from at least one of c_1 and c'_1 . Therefore, at least one of the two Λ -sequences is such that $\underline{\text{cho}}_2(b) \neq \underline{\text{cho}}_1(b)$, which is impossible if $\underline{\text{cho}}$ is objectively defined.

As proven by Goedel [1940], if set theory without the axiom of choice is consistent, then so is set theory with the axiom of choice. This suggests that there must be a way to interpret set theory so that the axiom of choice comes true. Indeed, we can do it by weakening the requirement of objectivity. The interpretation we have been discussing may be called the *many-user interpretation*. Objectively defined functions in this interpretation must not depend on the Λ -sequence that leads the user's way to truth, because for different users they may be different. But if there is only one subject of knowledge who uses the machinery of mathematics (or he does not care about other users), then there is only one Λ -sequence. This is the *one-user interpretation*. Then Theorem 10 does not apply, and it is easy to construct a choice function. For every uncountable set b we take the first implementation b_i of b which is not empty. Since b_i is a mechanical generator, we can uniquely pick up one of its elements and declare it $\underline{\text{cho}}(b)$.

Technically, this idea can be realized as follows. Modify the expression representing a set so that it becomes a pair: the

generator (as before) and one of its elements, if the set is not empty. When proving the regularity of a new candidate for Λ , pick up an element which is available with the current Λ_i ; make up the proper pair. The choice function will simply take the second member of the pair.

The many-user interpretation, in which the axiom of choice does not hold, seems more natural. But because of Goedel's result mentioned above, the interpretation of the axiom of choice is unimportant for the problem of consistency of set theory.

9. Uncountable Sets

The pow constructor stands alone from the other constructors we have defined. It calls the function lgs which provides access to the real-time process Λ representing our developing universe of discourse. If S is an infinite set, then there exists no generator which produces all the objects which can be produced by pow(S). This was first proven by Cantor, who interpreted it in the Platonist spirit as the evidence that pow(S) has "more" elements than S .

The notion of a hierarchy of static actual infinities is counterintuitive. Cantor's set theory introduced into mathematics a host of unimaginable entities, which later became being passed for the only "real" objects of mathematics. Yet in no reasonable sense do these entities exist, for we find them neither in reality nor in our intuition. The philosophical unsoundness of Cantor's theory has been recognized by many outstanding philosophers of mathematics starting with Henri Poincare who considered it as a perverse pathological condition that would one day be cured.

We interpret the mathematical formalism of set theory in terms of intuitively clear and unambiguous concepts. When Cantor proves that pow(S) has "more" elements than S , he only proves that whatever machine is offered to us as a generator or enumerator of the elements of pow(S), we always can construct a new element, not yet accounted for. These 'us' and 'we' are absolutely essential for the meaning of the proof, even if they are

avoided by using a different grammatical form. It is impossible to understand Cantor's proof without 'we always can'. It shows that the construct $\text{pow}(S)$ cannot be interpreted in terms of model-time processes only, but involves inextricably the idea of real time in which we live and in which 'we always can' create one more element.

The identification of sets with generators in constructive approaches to the foundation of mathematics usually stumbles over the interpretation of non-denumerable sets. In our theory, because of the introduction of metamechanical processes and the concept of objective interpretability, the constructive foundation is compatible with the existence of genuinely non-denumerable sets. We should discuss how this becomes possible.

Take a set S . If it is mechanical, the order in which the members are produced is objectively defined. A set S whose generator is a metamechanical process can be seen as the limit of the real-time sequence S_1, S_2, S_3, \dots etc., where S_i is the set generated by the mechanical generator corresponding to our knowledge at the i -th moment in time. Since the knowledge of the subject can only grow, each next set in this series includes all previous sets as subsets. We also can see S as the union of all the sets S_1, S_2, \dots etc. Can we enumerate the members of such a set? There is a way to do it, which has been used since Cantor, and is known as diagonalization. Arranging the members of all the sets in the infinite rectangular table

$$\begin{aligned} S_1 &= \langle a_1, a_2, a_3, a_4, \dots \rangle \\ S_2 &= \langle b_1, b_2, b_3, b_4, \dots \rangle \\ S_3 &= \langle c_1, c_2, c_3, c_4, \dots \rangle \\ S_4 &= \langle d_1, d_2, d_3, d_4, \dots \rangle \\ &\dots \end{aligned}$$

the diagonalization process generates all of them in the order:

$$a_1, a_2, b_1, a_3, b_2, c_1, a_4, \dots \text{ etc.}$$

When an element appears repeatedly, we ignore it, thus counting only the first entry. This allows to enumerate the union set S .

If there is a mechanical generator which generates the sequence S_1, S_2, \dots of mechanical generators, and the equality of the elements of the sets is tested by a finite mechanical procedure, then the diagonalization is also a mechanical procedure. A recursively-enumerable set of recursively-enumerable sets of objects is recursively-enumerable. If the elements of the sets are infinite sets themselves, we need a recourse to human knowledge to decide on their equality. This makes the diagonalization procedure not recursively-enumerable, but still leaves it objectively definable. We can now consider a more general case where the sets S_i are denumerable, and so is the set of the sets S_i . Then the diagonalization procedure will be objectively definable, and the union set S denumerable.

Consider, however, the case where the sets S_i are generated in real time by lgs. Although we can go on jumping from S_i to S_{i+1} infinitely, we can do it only in real time, and are unable to construct -- once and forever -- an objectively defined generator which produces all of them. Therefore, we cannot use diagonalization. Although the limit of \bigwedge_i for $i \rightarrow \infty$ (loosely understood) is the same for all subjects of knowledge, the specific stages in which it is achieved may be different. The order in which the interpretability of different expressions is proved may vary from subject to subject, it is not objectively defined. Therefore, the resulting enumeration is not objectively defined either. Moreover, even with a given sequence of S_i 's, the enumeration, which must be a process in real time, will give different results depending on the frequency with which this process reads current values of S . For example, if it makes readings twice as frequently as was assumed in the rectangular table above, then the table will be different, namely:

$$S_1 = \langle a_1, a_2, a_3, a_4, \dots \rangle$$

$$S_2 = \langle a_1, a_2, a_3, a_4, \dots \rangle$$

$$S_3 = \langle b_1, b_1, b_2, b_4, \dots \rangle$$

$$S_i = \langle b_1, b_2, b_3, b_4, \dots \rangle$$

...

Now the ordering given by diagonalization is different from what it was above; e.g., a_3 precedes b_1 , while they were in the opposite order before.

So, this is how non-denumerability is interpreted in our theory. Non-denumerable sets are generated by metamechanical processes, in the interaction between the user and the machine. The property of belonging to such a set is objectively definable. Since the only way a proposition of set theory can include a set is through the mediation of the predicate of membership ϵ , propositions may refer to non-denumerable sets and still be objectively defined. But the order in which the elements of a non-denumerable set are produced is not objectively defined; it depends on the user. There exists no objectively definable function which would map all elements of such a set on the set of whole numbers.

This interpretation is radically different from the one given by Cantor, according to which a non-denumerable set has "more" elements than a denumerable set. In our theory every set, and every element of every set, is represented by an expression. The set of all expressions (including uninterpretable) is, of course, denumerable (and even recursively-enumerable). Thus all infinite sets in our theory are intuitively perceived as having "no more" elements than the denumerable set of all expressions. The theorem that a subset of a denumerable set is denumerable remains, of course, true, but this does not lead to a contradiction, because the generator of all expressions is not a regular set. The fact that our universe of discourse stays always within the set of all expressions does not help us to enumerate it. By Theorem 3, it is impossible to separate interpretable expressions from uninterpretable, even if we are allowed to use metamechanical processes, not only mechanical. The "universe of universes" is undefinable. Meaningful objects and propositions can only be constructed inductively, from bottom up, in the interaction between the user and the machine. Period.

Acknowledgements

I appreciate discussions of the Cybernetic Foundations which I had with several colleagues: Karel Hrbacek and Michael Anshel of the City College, Angus Macintyre of Yale, Avgustin Tuzhilin of the College of Staten Island, and Martin Davis of the Courant Institute in New York.

REFERENCES

- Beth [1968] *The Foundation of Mathematics*
- Goedel, Kurt [1940] The consistency of the axiom of choice and the generalized continuum-hypothesis with the axioms of set theory, *Annals of Math. Studies*, No 3, Princeton, N.J.
- Orlov, Yuri F. [1978] Wave calculus based upon wave logic, *Intern. J. of Theor. Physics*, Vol 17, pp.585-598.
- Orlov, Yuri F. [1982] The wave logic of consciousness: a hypothesis, *Intern. J. of Theor. Physics*, vol 21, pp.37-52.
- Tarski, A. [1933] *The Concept of Truth in the Languages of Deductive Sciences*.
- Turchin, V.F. [1980] *The Language Refal*, Courant Computer Science Report # 20, New York University.

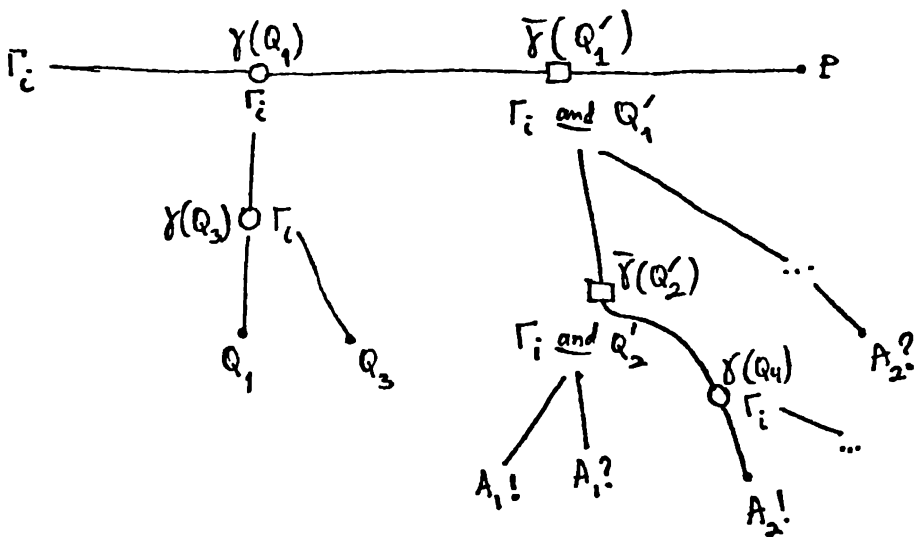


Fig. 6. Derivation tree

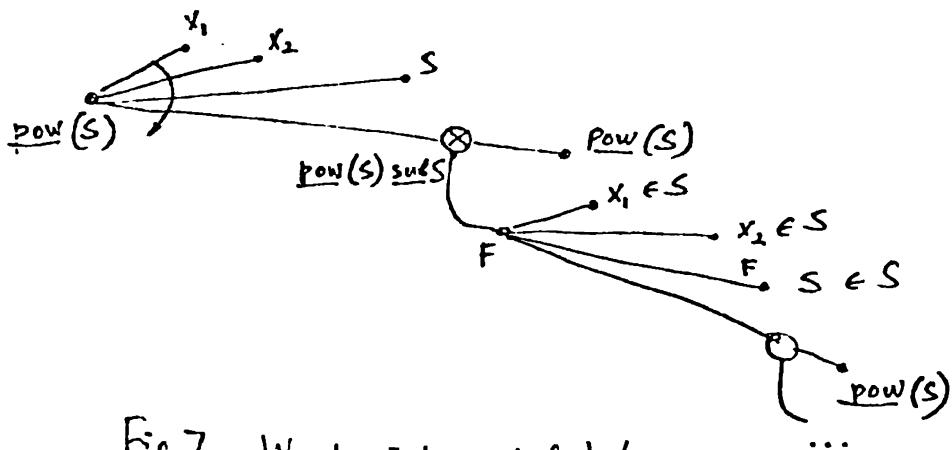


Fig. 7. Weak interpretability
of $\underline{\text{pow}}(S)$

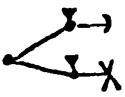


Fig. 1. Semantic graph for $A \& B!$

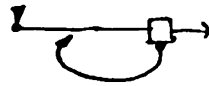


Fig. 3. Semantic graph for isz!

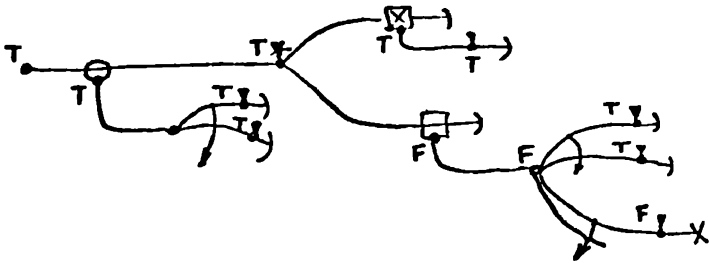


Fig. 2. Semantic graph for if $y(B?)!$ then $or(\bar{y}(A!), \bar{y}(A?))!$

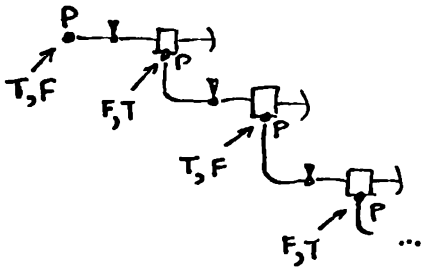


Fig. 4. The paradox of the liar

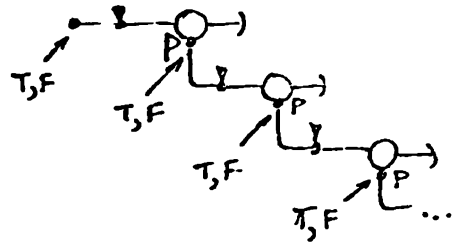


Fig. 5. The paradox of the saint