THE CYBERNETIC FOUNDATION OF MATHEMATICS

by

Valentin F. Turchin

The City College
of
The City University of New York

SCHOOL OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCES

PREFACE

This is a preprint of the book with the same title which has not yet taken its final form. Chapter 7 and the part of Chapter 6 dealing with the continuum hypothesis are not included because I am still working on them. I am also planning to include much more exercises than can be found in this edition.

I appreciate the discussions of the early version of parts of this book which I had with professors Karel Hrbacek and Michael Anshel of the City College of New York. I am much obliged to Susan Goeckel, Robert Nirenberg, and James Piccarello for their help in editing.

The work on this book was partly supported by the National Science Foundation grant # MCS-8007565.

ALL figures are at the end of the book. Symbol ▽ signifies the end of a subsection starting with a boldface specifier, like **Proof**, **Example**, etc.

C O N T E N T S

Chapter 5 LOGIC

Chapter 6 SET THEORY

CFMBB:
3. Functions

introduction

At the turn of this century there was a lot of talk about the crisis in the foundations of mathematics. Although no convincing answers have been found yet to the questions which arose then, the word 'crisis' is not used in that context any more. This is understandable: a crisis which lasts for almost one hundred years becomes perceived as a normal condition, not a crisis. A normal condition, however, is not necessarily a satisfactory one. The 'crisis' is still there. A vivid testimony to that is provided by the excellent picture of "the ideal mathematician" of the present day given by P.J.Davis and R.Hersh in their recent book *The Mathematical Experience*. The authors, well-known mathematicians themselves, call the hero of this picture 'ideal' not because he is perfect in any sense, but because he ideally represents his kind, or type. He is "the most mathematician-like mathematician". His imaginary field is "non-Riemannian hypersquares", and he pursues his studies with passionate devotion. He spends all his days in contemplating the non-Riemannian hypersquare. "His life is successful to the extent that he can discover new facts about it".

There are two aspects of the crisis in mathematics, and they are reflected in the picture of "the ideal mathematician": one concerns the nature and the very existence of the things he studies; the other is the reason he should study them. But let us give the floor to the authors.

> The objects which our mathematician studies were unknown before the twentieth century; most likely, they were unknown even thirty years ago. Today they are the chief interest in life for a few dozen (at most, a few hundred) of his comrades. He and his comrades do not doubt, however, that non-Riemannian hypersquares have a real existence as definite and objective as that of the Rock of Gibraltar or Halley's comet. In fact, the proof of the existence of non-Riemannian hypersquares is one of their main achievements, whereas the existence of the Rock of Gibraltar is very probable, but not rigorously proved.

> ...He rests his faith on rigorous proof; he believes that the difference between a correct proof and an incorrect one is an unmistakable and decisive difference... Yet he is able to give no coherent explanation of what is meant by rigour, or what is required to make a proof rigorous. In his own work the line between complete and incomplete proof is always somewhat fuzzy, and often controversial.

The authors summarize the discussion of the objects of mathematics in the following way.

> Mathematicians know that they are studying an objective reality. To an outsider, they seem to be engaged in an esoteric communion with themselves and a small clique of friends. How could we as mathematicians prove to a skeptical outsider that out theorems have meaning in the world outside our fraternity?
>
> If such a person accepts our discipline, and goes through two or three years of graduate studies in mathematics, he absorbs our way of thinking, and is no longer a critical outsider he once was. In the same way, a critic of Scientology who underwent several years of "study" under "recognized authorities" in Scientology might well emerge a believer instead of a critic.
>
> If the student is unable to absorb our way of thinking, we flunk him out, of course. If he gets through our obstacle course and then decides that our arguments are unclear or incorrect, we dismiss him as a crank, crackpot, or misfit.
>
> Of course, none of this proves that we are not correct in our self-perception that we have a reliable method for discovering objective truths. But we must pose to realize that, outside our coterie, much of what we do is incomprehensible. There is no way we could convince a self-confident skeptic that the things we are talking about make sense, let alone "exist".

The failure to convince the skeptic is not simply a result of the complexity of mathematical constructs. One need not know in detail the construction of a machine to understand what the machine is doing. No computer scientist would second the mathematician's complaint of incomprehensibility, although the complexity of big computer systems created by dozens of people leaves far behind the information content of the theory of "non-Riemannian hypersquares" or the likes of it. Also, such a complaint could

hardly have been made before the end of the nineteenth century. Should "the self-confident skeptic" say that he does not understand what are numbers and geometric figures, he would be simply sent to hell, and with good reason.

I believe that mathematics is incomprehensible for outsiders because it is incomprehensible for the mathematicians themselves; otherwise they would be able to explain at least its base. But it is just in the base where the trouble is. Contemporary mathematics is based on set theory, which deals with entities that defy comprehension. Yet the objects of mathematics, though built on the set-theoretical base, convey to everyone who is studying them the feeling that he is dealing with "a real thing". Paradoxically, this feeling is shared also by those mathematicians who specialize in set theory itself. There can be only one explanation of this paradox, or at least no other can be immediately seen. It is that the formalism of set theory *does* refer to some reality, which is -- as a reality -- quite comprehensible, while the present interpretation of this formalism based on the concept of actual infinity is not only incomprehensible, but simply wrong. It is conceivable that if this really is the case, the mathematicians could have developed their set-theoretical intuition in response to the real, and not the proclaimed, objects of set theory.

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

while we habitually entertain the idea that in set theory we can express everything that can be subject to rigorous mathematical treatment. A theory based on these concepts must be equally 'strange'. To use an expression popular among physicists and coined by Niels Bohr, such a theory must be 'crazy enough'.

Such a 'crazy' theory is developed in the present book. It leads to a full acceptance of the formalism of set theory, but interprets it in the agreement with the principles of constructivism, using only the idea of potential, but not actual, infinity. Our theory of mathematics has the following features.

(1) Mathematics is seen as a branch of science. The objects of mathematical knowledge are of the same nature as those of other sciences: the abstracted phenomena of the world we live in. As every branch of science, mathematics has its own type of objects, and this may lead to significant differences of quantitative character. However, there is no difference of principle with regard to the nature, method of acquisition and reliability of mathematical knowledge as compared to the knowledge of natural sciences.

(2) In ageement with the contemporary philosophy of science, the meaning of a mathematical (as well as any other) proposition is defined as our ability to use it as a generator of verifiable predictions about real world processes. If a proposition cannot be interpreted as a generator of predictions, it is meaningless and has no place in our theory.

(3) Our theory is concerned with *symbolic* mathematics, which is the official language of all mathematicians nowadays. While the importance of three-dimensional geometric intuition is acknowledged, no attempt is made to analyze its role.

(4) In addition to mechanical processes of computation and proof, which play so important a role in contemporary mathematics, we introduce *metamechanical* processes, which cannot be modeled in such devices as Turing machines, or defined through recursive functions. Metamechanical processes differ from mechanical processes not by the kind of machinery they use, but in that they are controled by the *subject* of knowledge, i.e. its owner and developer. In the conceptual apparatus of modern physics, the inseparability of the phenomenon and the observer, i.e. of the object and the subjects of knowledge, is well known. Our

theory introduces it in mathematics.

(5) With this conceptual toolbag we interpret the fundamental aspects of mathematics, in which we also include formal logic. We show that there are two ways to define the concept of truth in our formalism. One leads to intuitionistic logic, the other to classical logic. Goedel's theorem is shown to be of crucial importance for the acceptance of the law of the excluded middle. Our theory stresses the positive aspect of Goedel's theorem.

(6) Set theory is interpreted and the axioms of Zermelo-Frenkel are proven as theorems. No special effort is made to avoid paradoxes. They simply do not appear, which is one of the results of our definition of the interpretable proposition. Sets are interpreted as processes which generate objects. Metamechanical processes provide for not recursively-enumerable and uncountable sets.

(7) The consistency of set theory is proved. This does not contradict Goedel's result of the impossibility of proving the consistency of a theory in the theory itself, because our theory cannot be formalized in set theory. The continuum hypothesis is also proved.

(8) The basic, all-pervading methodological principle of our approach is the concept of *metasystem transition*. The technical means we use to formalize this idea is the language *Refal*. Our formalism is designed so as to avoid explosive increase in volume when translating mathematical propositions. Refal is implemented on computers and used as a programming language for writing complicated symbol-manipulation programs. At the present time, the machines introduced in mathematics for the purpose of formalization are used only for mental experiments, as theoretical devices. One of our goals is to bridge the gap between mathematics and computer science by formalizing mathematics in such a way that the mechanical processes referred to in its definitions could be actually run on a computer.

Let us return to "the ideal mathematician" created by Davis and Hersh. They give a remarkable description of the way the mathematician works. It deserves a long citation.

> The ideal mathematician's work is intelligible only to a small group of specialists, numbering a few dozen or at most a

few hundred. The group has existed only for a few decades, and there is every possibility that it may become extinct in another few decades...

He finds it difficult to establish meaningful conversation with that large portion of humanity that has never heard of a non-Riemannian hypersquare. This creates grave difficulties for him; there are two colleagues in his department who know something about non-Riemannian hypersquares, but one of them is on sabbatical, and the other is much more interested in non-Eulerian semirings. He goes to conferences, and on summer visits to colleagues, to meet people who talk his language, who can appreciate his work and whose recognition, approval, and admiration are the only meaningful rewards he can ever hope for.

At the conferences, the principal topic is usually "the decision problem" (or perhaps "the construction problem" or "the classification problem") for non-Riemannian hypersquares. This problem was first stated by Professor Nameless, the founder of the theory of non-Riemannian hypersquares. It is important because Professor Nameless stated it and gave a partial solution which, unfortunately, no one but Professor Nameless was ever able to understand. Since Professor Nameless' day, all the best non-Riemannian hypersquarers have worked on the problem, obtaining many partial results. Thus the problem has acquired great prestige.

When speaking with fellow-hypersquarers the ideal mathematician uses an informal jargon, but the style of his published writing is different.

...There he piles up formalism on top of formalism. Three pages of definitions are are followed by seven lemmas and,finally, a theorem whose hypotheses take half a page to state, while its proof reduces essentially to "Apply Lemmas 1-7 to definitions A-H."

His writing follows an unbreakable convention: to conceal any sign that the author or the intended reader is a human being. It gives the impression that, from the stated definitions, the desired results follow infallibly by a purely mechanical procedure. In fact, no computing machine has ever been built that could accept his definitions as inputs. To read his proofs, one must be privy to a whole subculture of motivations, standard

> arguments and examples, habits of thought and agreed-upon modes of reasoning... If (heaven forbid) the fraternity of non-Riemannian hypersquarers should ever die out, our hero's writings would become less translatable than those of the Maya.

A series of conversations follows between the ideal mathematician and people of different professions. A public information officer of the University asks him what possible applications his research may have. Here is the answer of the mathematician and a portion of the subsequent dialogue.

> *I.M.*: I've been told that some attempts have been made to use non-Riemannian hypersquares as models for elementary particles in nuclear physics. I don't know if any progress was made.
>
> ...
>
> *P.I.O.*: Do you see any way that the work in your area could lead to anything that would be understandable to the ordinary citizen of this country?
>
> *I.M.*: No.
>
> *P.I.O.*: How about engineers or scientists?
>
> *I.M.*: I doubt it very much.
>
> *P.I.O.*: Among pure mathematicians, would the majority be interested in or acquainted with your work?
>
> *I.M.*: No, it would be a small minority.

It is enough. To give your life to the non-Riemannian hypersquare only because Professor Nameless defined the concept and proved a theorem which nobody except him could understand, this is not an appealing prospect to a young man. With all the beauty and depth of contemporary mathematics, the picture of the ideal mathematician is rather depressing. It seems that something is missing in mathematics: the top of the hierarchy of goals. There is no general strategy of mathematics in sight, such as we clearly see in physics,biology, or computer science. There is no approach or conception which would not simply indicate the place of mathematics in the contemporary world -- the place that is secured by the achievements of our predecessors -- but would indicate where and how we should go on. "The ideal mathematician" reminds one of a week offspring of a once strong family who is rich in heritage but poor in his own spirit, and completely absorbed by minor family affairs.

I believe that this situation is a direct consequence of the crisis of foundations. Indeed, how can we hope to find a lead in

constructing mathematics if for one hundred years there has been no understanding on what mathematics is about? Conversely, we can hope that a new approach to the foundations will provide at least some new guidance to tell the important from the unimportant in mathematics.

Other sciences, notably physics, are dependent on mathematics for the models they can use to construct their own models of reality. Quantum mechanics could emerge because the physicists discerned in some mathematical structures, which were created before and with a different purpose, the features they could use for the descripton of observed facts. This can serve as an argument for "zoological mathematics", that is for the study of all sorts of mathematical creatures in anticipation of their possible use. But if one remembers the huge, potentially infinite, volume of this "fauna", it will become clear that we can hardly hope for success without any guide, without a theory which could tell us how to create useful structures. But such a theory presupposes a convincing theory of mathematics.

According to the present formalistic philosophy, mathematics is a language without semantics. This philosophy reduces all mathematics to just the axiomatic method. You give me your axioms, says the mathematician, and I shall deduce as much from them as I can. I do not know and do not want to know what your axioms mean. For me they have no meaning: just formal objects to manipulate.

Davis and Hersh express strong dissatisfaction with the formalistic philosophy of mathematics as contradicting the belief of the working mathematician in the objective existence of his objects, and downgrading, if not damaging, his perception of these objects. My impression is that even though some mathematicians may resent the portrait of their "ideal" colleague, the general attitude with regard to the present condition of mathematics expressed by Davis and Hersh is going to be shared by more and more people. This is why I found it important to discuss "the ideal mathematician" at length.

The axiomatic method is a great invention, but it is not everything; moreover, in the present situation it is, probably, not the most important thing. The axiomatic method is good for studying, from different angles, a given mathematical object. But the question now is: what mathematical objects do we need? The

most important and creative part of the job needed today by natural sciences, by physics for example, is left hanging in the air between the physicist and the mathematician. Theories we create are based on formalizations of our intuitive and informal concepts pertinent to the studied phenomenon into formal mathematical models. But very often we have no, or not enough, pertinent intuitive concepts to construct a mathematical model. It may happen because our concepts are too fractional, and models based on them become too complicated and unilluminating. This is a frequent case in biology and social sciences. Or we may simply have no valid intuitive concepts about phenomena; this happens in the physics of elementary particles, where our macroscopic intuition is more misguiding than guiding. Then who will formulate the axioms and, before that, create the formal objects the axioms are about? If the axioms are given, will the mathematician really be able to work with them without understanding their meaning and basing his effort only on the formal rules of inference? It is more than doubtful, especially if you consider the very limited, not to say miserable, successes of computer theorem proving up to date.

Our approach to mathematics is semantic. We treat the language of mathematics as having a definite, precise and unique meaning and speaking about real observable phenomena. The concept of a *process*, on which our semantics hinges, bridges mathematics and natural sciences, because it is applicable both to *linguistic processes* of mathematics, and to natural phenomena. I hope, therefore, that this theory will help find new approaches to the most general and important problem of contemporary science: how to construct mathematical structures necessary for successful theories of natural phenomena? What we need is a *metatheory* of scientific theories.

As to the internal needs of mathematics and compter science, a quite definite program follows from the theory; it is briefly outlined in Chapter 7. With our philosophical approach, mathematics and computer science share the same conceptual foundation. At present, a wide gap separates the foundations of mathematics and those of computer science. A theory which essentially unites them into one scientific discipline should be for their mutual benefit.

CHAPTER 1

The Nature of Mathematics

1. The reflection theory

The first and very long-lived philosophy of mathematics known to us was elaborated, as many other firsts in philosophy, by Plato. Mathematics is understood by Plato in the framework of his *theory of ideas*. According to this theory, the things we see around us and perceive otherwise do not represent the ultimate rteality of the world. They are sort of reflections, or manifestations, or shadows of something else, namely *the abstract ideas* of things. It is these ideas that constitute the ultimate reality and make the observable material things possible. Material things are unstable and imperfect, they come and go. Ideas are unchangeable and eternal. While we know material things by means of sense organs, ideas are perceived by our mind through the process of reasoning.

The first reaction of a modern man when exposed to this theory is usually: what a nonesense! How does he know about these ideas? Is not it a pure invention, a fantasy? The second reaction would be: how strange that people could take this stuff seriously for two thousand years. Was it a collective self-hypnosis? It seems incomprehensible.

But if we care to give it a second thought, we find that far from being a strange aberration, Platonism is an inevitable stage in the development of philosophy. First of all, we should not be confused with the use of the word 'idea' in Plato's theory. It has nothing to do with 'thought'. Plato is no subjectivist or spiritualist. His 'idea' of a thing must be understood rather as the pattern, or form, to be discerned in it; or, as we would say in todays language, its *organization*. Plato's dichotomy between things and ideas corresponds to the dichotomy in our language between energy-matter and information.

How do we come to Platonism? We start thinking about our thinking. We see that to express our thoughts we use language, therefore thinking about our thinking we also think about language; we are dealing with a thought-language complex. The usefulness of this complex, as everyone knows, is that it gives us a sort of copy, or a reflection, of reality. We say "Socrates", and this is only a word, but it has a meaning because it corresponds to the real person Socrates. Also, when I say "give me that apple", you will understand me if and only if you can see the real, material apple which corresponds to the word "apple" I have used. Sometimes we lie or deceive ourselves. I can say or think that I am holding an apple, while in fact it is a rock. In such cases there is no correspondence between the thing and th thought or language. So, the first step of philosophizing lead us to what is known as *the theory of reflection*. It says that the meaning and the significance of thought-language is in its possible correspondence with the reality. If this correspondence takes place, the thought is true, otherwise it is false.

If you accept the theory of reflection, and people usually do, you come immediately to Platonism. Our language includes not only those words which are in a direct one-to-one correspondence with specific material objects, like "Socrates", or "the apple I am now eating", but also general or abstract concepts: "a man", "an apple", "a triangle". We face a dilemma now. Either we throw away the greater part of the language as meaningless and leave only proper names and concrete propositions, or we have to admit that there is some reality standing behind our abstract notions. Hardly anybody would seriously contemplate the first alternative. Then only the second is left, and this is Platonism. If the existence of the second type of reality is accepted, it immediately becomes primary, and the first, empiric reality -- secondary. Because you can kill Socrates and destroy a house, but you cannot kill the concept of a man or the idea of a house. You can eliminate every triangular thing you can reach, but in doing that you do not eliminate *the triangle* as such. Knowing what the triangle is, you can again create a lot of triangular things. At the same time, when you create those things you do not create *the triangle*. It has always existed and will exist forever.

In the above passage, the reader could have noticed that when I used the abstract triangle as an example of an idea to have an independent existence, the argument became more persua-

sive than when using the house or the apple. Mathematics has always played an important role in Platonism, supplying it with the best examples and arguments. It is impossible to simply say that the mathematicians deal with non-existent objects. Yet it is clear that the number two and the equilateral triangle do not exist in the same way as rocks, apples and houses. They exist as ideas. Therefore, ideas do exist. Without a further and deeper analysis of thinking, Platonism is inevitable.

Philosophers, of course, went on with their analysis. Difficulties and inconsistencies were discovered (mostly at the interface between the world of things and the world of ideas), skeptics announced that they were not convinced, and heated debates abounded. The heat, of course, came not because of mathematics, but because of religion. For not only numbers and triangles are abstract notions, but also good and evil, justice, grace, and God. It was, and still is, very important for people to know if these things really exist.

The angle from which we look at the history of philosophy in this very quick review is the theory of reflection and its significance for the concept of truth in mathematics. Our next station will be with Immanuel Kant. We want only a brief summary of what happened to the theory of reflection during the two millenia between Plato and Kant.

Plato's doctrine was challenged in two planes: ontological and epistemological. In the ontological plane the debate was between the *realists*, or Platonists, and the *nominalists*; the problem was known as the Problem of Universals: do the Universals, i.e. the entities corresponding to general, abstract concepts, really exist? Those who contended, with Plato, that they really exist were called realists. Now, since the word 'realist' acquired later another meaning, this party in the debate is usually called Platonists. The other party asserted that Universals were only words, *names*, to denote all those real things that qualified according to the meaning of the concept. 'The house' does not exist as such, it is only a name applicable to all specific houses. (At last I see people who sound reasonable, the modern man will say). But mathematics was a stumbling block for consistent nominalism. 'The house' may be only a name, a symbol, and have no existence of its own. We could replace it by another name, e.g. 'maison'. But what about the objects of mathematics? If they are only names which can be arbitrarily chosen and

changed, how can mathematicians discover their properties and prove theorems? The triangle of mathematics cannot be identified with the set of all material objects of triangular form, because we have no exact triangles in nature. Mathematics proves that the sum of the angles of any triangle is 180 degrees. But if we measusre the sum of the angles of any real triangular object, it will be close but never exactly 180°.

In the epistemological plane, the opposite trends of thought were rationalism and empiricism; the problem was: what is the reliable and preferable source of our knowledge? For rationalists this source was our reason. Empirical data are chaotic, there can be no necessity in them, necessity is to be found in the world of ideas only. We use our sensual experience as the source of suggestions for our reason to look in certain directions, and to test and try our conclusions, but a proposition can be cosidered as necessarily true only if it is derived by our reason from some primary, fundamental, self-evident truths perceived as necessary in the world of ideas directly by the mind. Thus the ultimate source of true knowledge is reason.

Nothing of that kind, said empiricists. All we know is learnt through our sense organs. The soul of a newborn baby is a clean slate (Locke: *tabula rasa*) into which his experience writes down the story. Our ideas are only a reflection, usually and most usefully a condensed one, of our experience. The ultimate source of knowledge is our sensual experience. The knowledge which is not rooted in experience is not a knowledge but a fiction, a fantasy.

By the end of the eighteenth century, a working compromise was established between the two camps. Natural sciences, which made tremendous success and compelled the old philosophy to retreat, became the province of nominalism and empiricism. Mathematics and religion were retained by Platonism and rationalism. The common ground for both camps, on which there was no fighting was the theory of reflection. It was admitted now that there are different types of knowledge. But no matter what the source of knowledge and the nature of its objects, the idea that knowledge is a correspondence between thought and object seemed self-evident.

The downfall of the reflection theory started with Kant. Compromise is good for politicians but not for philosophers. Kant

looked for an organic synthesis of empiricism and rationalism which could explain the success of empirical science, leave place for mathematics, and provide a sound basis for ethics and religion.

The essence of Kant's message is this. (Let me stress the word *message*. I am not trying to expose the whole of Kant's philosophy in its actual terminology. It is rather a contemporary reading of Kant).

Both empiricism and rationalism are in error when they think that they can discuss or even think about the relation between the thought and its ultimate object. Suppose I want to compare my idea of an apple and this apple as it really is. But what does it mean 'as it really is'? When we come to think about it, we see that 'the apple as it really is' is again my idea of the apple. Things we perceive, as Berkeley eloquently argued, are given to us only in our own sensations, only as objects of our thought. We do not know and have no means to know what are things in themselves, independently of our perception.

Let us analyze how the reflection theory comes into being. When we start to think about our thinking and our language, we notice, quite correctly, that our ideas reflect, or correspond to, or refer to, some reality. (If they do not, in an obvious way, a psychiatrist should be consulted). So my thought of an apple, represented in cell *b* in Fig. 1.1, reflects the real apple of cell *a*. How do I know it? Well, from my observation of other people and reasoning by analogy. My friend professor Nameless sees an apple, takes it, bites and chews. He certainly does not mistake it for a cigaret or a bus. Neither do I, I am sure. I also can analyze my thinking about my idea of an apple, which is represented in cell *c*. Again, it reflects something, namely the reality of cell *b*. I can go up and up in this hierarchy, but I hardly need it. Much more interesting is to go down. I note that if somebody looked at me -- or at us, the collective subject of human knowledge, humanity -- then he would see that what I accept for an apple in cell *a*, is only my perception or idea. What does it reflect? I denote the missing cell *x* and start thinking about it. It seems obvious that *x* is the real apple, which is reflected by my idea of the apple. This is what the reflection theory states: our ideas of things are reflections of things as they really are.

When we look closer, however, we see that 'the thing as it really is' is a mirage, like a spurious sun. When I look at myself from outside. I put in correspondence to every cell *y* the cell 'my thought of *y*', as shown in the second column in Fig.1. I say now: what I take for *a* is really *b*'; what I take for *b*, is really *c*', etc. It seems to me that I found the answer to the question what is *x*: it is really *a*'. But in fact *a*' is identical to *a*. Cell *x* is only a position in the diagram devoid of any contents. When we speak about *x*, we imply that there is some contents in it, but this contents is *a*. The reflection theory makes a copy of *a*, i.e. of our perception of a thing, and sells it for a new item *x*, the thing as it really is. Kant's thing-in-itself is the position *x* in the diagram, but not a meaningful part of speech. It should play no role in our reasoning.

Knowledge *is not* the correspondence between the thing and the thought, because we never compare the thought and the thing. We compare only one thought to another. But what is knowledge then? Is it simply a chaotic collection of the data of our sense organs? Not at all. Simplest analysis will show that our ideas are *organized* in a certain way. This organization is the work of reason. We can distinguish three levels in thought: sensations, perceptions, conceptions.

On the first, basic level we find the raw material of the mind, our feelings, sensations: light flashes, coloured spots, the sound of somebody yelling, the warmth of the milk in the mouth, the touch of cold stone, etc. This is the experience of an infant in the first days of his life. It does not yet constitute knowledge.

There must be a tremendous work done to organize the primary stream of sensations into our perception of distinct objects situated in space and changing in time. This work is done by our mind and it produces the second level of thought, the level of perceptions. Space and time, according to Kant, are not things-in-themselves, they are *forms of perception*.

On the third level, the peceptions organized in space and time serve as the material to be processed into still higher type of thought, conceptions. The forms used by the mind to mold conceptions from perceptions are called by Kant *categories*. The idea of *cause*, for instance, is one of categories we use in forming conceptions. There is no causality in the sensations we

get from the external world; it is our way to organize the sensations.

What is the the goal of the philosopher? What can he do and what he cannot? He certainly cannot know anything about "things as they really are", their "real nature". Knowledge is not a reflection in the subject of knowledge of the object of knowledge. Knowledge is their interaction. The subject and the object are inseparable in knowledge. When we try to consider the object of knowledge in isolation from the subject, we leave no place for knowledge. The only task a philosopher can set to himself is *the critical analysis* of our knowledge on all three levels: sensations, perceptions, conceptions. By this analysis we can distinguish what comes from the raw material of sensations from what is introduced by the mind in processing this material. The empiric and the rational are not two opposing sources or methods of knowledge; they are two aspects of the same phenomenon.

Kant divides judgements (propositions) into *analytic* and *synthetic*. The analytic judgements are those in which the contents of the judgement, the logical predicate, only expresses what must be there by the definition of the logical subject of the judgement. When you say, for instance, that the apple is a fruit, this is an analytcal judgement. Essentially, analytical judgements are definitions, which can be made *a priori*, without any reference to experience. To put it in the terms of the present time, they carry no information. Synthetic judgements carry information, because they link things which are not linked by their definition. 'Jack killed a wolf' is a synthetic judgement. We can state that Jack killed a wolf only after having the corresponding experience, *a posteriori*.

As in the case of Plato, mathematics plays an important role in Kant's philosophy. Mathematics, says Kant, contains a lot of synthetic judgements. For instance, the proposition that the sum of the angles of a triangle is 180 degrees is synthetic, because the triangle is defined as a figure of three sides and three angles, but nowhere in the definition do we find that the sum of the angles must be 180°. Yet this judgement is made without any reference to empiric facts, *a priori*? How can it be possible? How can synthetic judgements be possible a priori?

Kant's answer is: because they have nothing to do with sensations, but reflect the way our mind *processes* sensations.

Mathematics is about the forms which our reason applies to sensations in order to mold them into perceptions. This is why mathematics is so important and useful on the one hand, and does not depend on empiric data and our experience, on the other hand. The example of mathematics shows that the pure forms of our reason can be subject to a detailed analysis. The purpose of Kant's main work, *The Critique of Pure Reason*, was to explore the limits of such an analysis with respect to the traditional problems of metaphysics.

The rationalist trend of thought was first to realize the importance of Kant's synthesis, and draw conclusions from it. The result is known as German idealism. If object and subject are inseparable in knowledge, and our knowledge is the only reality we have, then does it not suggest that the subjective element, the quality which makes it possible to be the subject of knowledge, is at least as important a constituent of reality as the objective element, the quality which is common to all material things we perceive? This is the starting point of idealism. The forms which our mind applies to sense experience, in particular the laws of logic, reflect (the reflection theory again!) the deepest laws of All That Exists. From the ancient Greeks the law of contradiction, i.e. the principle that a proposition is proved if its negation is shown to be contradictory, was considered the most fundamental logical law which allows one to discover those truths which are necessary, but not immediately obvious. Hegel, the most influential German idealist, transforms this law into the universal driving force of the developing world.

During one hundred years after Kant, the empiricist trend of thought ignored the work of the Koenigsberg philosopher. This was the time of the triumph of science, especially of Newtonian mechanics and its applications. The scientists did not need Kant; the reflection theory and the compromise between empiricism and rationalism kept on working excellently. The only noticeable change was with respect to hypothesis. Newton believed that he managed to discover the true and absolute laws of mechanics, which cannot be renounced or modified any more than the laws of geometry. He draw a sharp distinction between a discovery and a hypothesis, and did not see much sense in the latter. "Hypotheses non fingo", he said: I do not feign hypotheses. Later, however, the scientists found it necessary to "feign" hypotheses. The method of science became described as *hypthetico-deductive*. A scientific theory is always created as a hypothesis, and then is

tested against experimental facts. However, if it is well-tested, then, according to the views of the 19th century scientists, the theory still was to be considered as a *discovery*. This word shows that the contents of the theory "is there" as an objective reality, an objective law of nature, and scientists only express it in terms of some -- usually mathematical -- language. We see here the reflection theory in its full. Not only do the *nouns* of the scientific language correspond to real material particles and their conglomerates, but the *sentences* of that language also reflect the objectively existing natural laws.

This view of scientific knowledge is formally independent from the ontology you assume; in practice, however, it strongly prefers the materialistic worldview, according to which the ultimate reality of the world is matter. This term may be understood with different degrees of sophistication, but its general tendency is to move the ultimate reality away from the subject of knowledge or action and closer to its object.

Only in the very end of the 19th century the importance of Kant's message for science was recognized, and only by a few far-seeing philosophers and scientists. Probably, the single most important factor which led to the formation of a new trend of thought was the discovery of non-Euclidean geometry and the related tendency toward axiomatization of mathemtics. For two thousand years the view of geometry as the only possible and unshakably true theory of spatial relations was one of sustaining pillars for the belief in human ability to discover the truth. It turned out now, that one of the axioms of geometry (i.e. geometry as given by Euclid) can be replaced by another axiom, which is diametrically opposite to the original axiom, and yet this does not lead either to internal contradiction in the theory, or to a contradiction with our sense experience. What is left then of the necessity and the objectivity of mathematical truths? Which of the two geometries is true?

Neither, answers Henri Poincare in his *Science and Hypothesis*. But not because both are false. Geometry cannot be true or false. It tells us nothing about real events in the world, it only provides us with a conceptual scheme to describe events. With respect to such schemes the concept of being true or false is simply out of place; a scheme can be more or less applicable, depending on the circumstances, but it can be neither 'true', nor 'false'. Suppose we measured the sum of the angles of a gigantic

triangle formed by light rays. According to the Euclidian geometry it must be exactly 180°, according to the Lobachevsky geometry, less than that. Suppose we found that the sum is less than 180°. Does it prove that Euclidian geometry is false? Not in the least. We could hold on to it as true but conclude that the sides of the triangle are not straight lines. In these conditions, we might say, light propagates along some curved lines, maybe arcs of some sphere.

This is a Kantian approach taken one step further. Mathematics is still considered as dealing with forms or schemes which we introduce in order to organize our sense experience. But for Kant these forms were immanent to our reason and therefore given and immutable, in a sense, objective. But Poincare sees two geometries in front of him, and knows that he can choose between them, or use them both, by his own accord. Ernst Mach analyzes the foundations of Newton's mechanics. The parallel between mechanics and geometry, which gave grounds for Newton to consider his creation as necessarily true, now works against him. If there are two geometries possible, then why not two or more mechanics? Richard Avenarius authored a philosophy known as *empiriocriticism*. Mach and Avenarius laid down the foundation of a philosophy of science which can be characterized as 'post-Kantian empiricism'.

Using this term, I have in mind not one definite philosophical system or school, but a family of such, with the most important epistemological features in common. Like earlier empiricism, post-Kantian empiricism accepts only sensations for the source of knowledge, and rejects Kant's idea of extracting from intuition some pure and immutable, transcendental forms of knowledge. Together with Kant, it rejects the reflection theory as naive and uncritical, maintains that the subject and object are inseparable in knowledge, and seeks to analyse critically their relationship. When we perceive a material object, our perception is not a reflection of a 'material object as it really is', but only a complex of sensations. When we create a theory which is amply corroborated by experiments, it is not a reflection of an 'objective law of nature', but a way to organize and foresee our sensations. The philosopher, or a thoughtful scientist, critically analyzes the stream of sensations we get from our sense organs and the way we form concepts to organize it. The meaning of these concepts is determined by the way they are translated into verifiable (in terms of sensations) facts. The concepts which cannot

be translated into sensations are meaningless, spurious.

In America, Charles Peirce, the founder of the school of philosophical *pragmatism*, associated the meaning of a concept or a theory with the consequences that result from the application of this concept or theory. In particular,a theory is true if it allows us to achieve our goals, if it "works in practice". This is close enough to the principle of verification. Pragmatism is also a variety of post-Kantian empiricism. In fact, we can reckon among the members of the same family all the most influential trends in the philosophy of science of the present time. To discuss these philosophies is not our intention.But it is important for our goals to discuss the relation between the post-Kantian empiricism and the discoveries of the physics of the 20th century.

A remarkable thing happened to post-Kantian empiricism: its conclusions, which resulted from pure philosophical reasoning seemingly unrelated to the current immediate needs of science, turned out to be of vital importance for the further development of physics. Albert Einstein, who like many other physicists of the turn of this century was deeply impressed by Mach's analysis, created in 1905 his (special) theory of relativity and produced a revolution in physics. He did that without writing a lot of formulas. The core of his theory was an analysis, in Mach's style, of some fundamental concepts of physics, primarily the concept of simultaneity. For our common sense it seems absolutely obvious that if two events occur at the same time, then it is "an objective fact", which cannot depend on whether or not you and I know about it, or whether we measure the times, and if we do then what reference system we use. It seems so obvious to our intuition that there is no need to check it and no possibility to avoid it. But Einstein discovered that the independence of the speed of light from the movement of the source, which had been firmly established as an experimental fact, can be explained if we abandon this postulate. He analyzed what it actually means, *in terms of observable facts*, for two events to be simultaneous. This analysis led him to a theory which was consistent with all experimental facts, but treated simultaneity as a relative phenomenon, dependent on the system of reference. Two events as perceived by one observer could occur at the same time, while in the perception of another observer one event would occur after the other. Acting on the primacy of observable facts over preconceived ideas, even when intuitively trustworthy, Einstein accep-

ted the constancy of the speed of light and rejected the absolute simultaneity.

In quantum mechanics, the physicists went even further in the decomposition of the "reality" of the mechanistic worldview into elementary obsevable facts. Quantum mechanics deals with material particles. But if we see these particles the way classical mechanics sees them, that is as tiny balls moving in the three-dimensional space along definite trajectories, then we immediately come to contradiction with experimental facts. A quantum-mechanical particle, say an electron, cannot have a definite position and a definite velocity at the same time. Wait a minute, a naive realist will say. You probable want to say that one cannot *measure* the position and the velocity of the electron simultaneously. But no, it is worse than that. If we assume that *in reality*, the electron is moving along a definite trajectory, then even admitting that there is no way to measure the position and the velocity simultaneously we still come to unresolvable paradoxes and contradictions with experiment. Maybe the electron is an amount of stuff distributed in space? No, this idea does not work either. Maybe the electron is a wave motion, like a sound or electromagnetic wave? No. We do describe the state of the electron by its *wave function*, but this function does not represent a distribution of energy-matter over the space, it is rather a *probability wave*, which represents our *knowledge* of the electron. When our knowledge changes abruptly as a result of measurement, so does the wave function.

Then what *is* this electron? Does it exist at all, or is it a pure fiction?

It was not easy to accept the idea that an electron is a material body, like an apple, but cannot be described in terms of the usual space-time concepts. It would be easier to think that 'actually' it still moves in a definite way, but we simply cannot know its trajectory. It takes some effort to recognize that what we want as 'the real electron' would not be 'the real electron', but just another model, a conception. If it could be used to predict observable facts, then it would be justified. Otherwise it is a fiction. In the case of the electron of modern physics, the picture of a small ball, if it comes with all its common sense details is a fiction.

It was still more difficult to accept the absense of ulti-

mate causality in the microworld. For centuries science held the view that all events have their ultimate causes, which can be in principle, if not in fact, discovered. This view was found wrong in the 20th century.

So, does this strange electron really exist, in the sense apples and other macroscopic objects exist? Of course it does. If the apple is an objectively existing material object, whatever precise meaning may be assigned to this statement, then the electron exists too. There is an important difference between them, yet it is not a difference of principle. Using Kant's terms, a bit too freely prhaps, the apple is a perception while the electron is a conception. But both result from our contact with the world through sense organs and a very considerable amount of brainwork. In the case of the apple, the work is done mostly using inborn features of sense perception, which are hereditarily determined and hardly changeable. In the case of the electron a very important part of the job is done using language; this type of work is an invention of human culture, and the concepts thus created change from time to time. The quantitative differences between the two cases with respect to brainwork are very significant, but with respect to such aspects as existence or objectivity there is no qualitative difference between electrons and apples.

The reason why the physicists of the 20th century had to reconsider radically their philosophy of nature is that they intruded into such spheres where their means of exploration became a significant part of the phenomenon. In the case of relativity theory, it is light, which can no longer be considered an instantaneous signal when the distances and the speeds became very great. In the case of quantum mechanics, it is collisions between elementary particles and their interaction with -- again -- light, but this time on a very small scale. European philosophy came to reject the reflection theory in a purely speculative way. Then its conclusions were confirmed by science. In retrospect, we can see that this confirmation had to arrive sooner or later. The reflection theory can picture 'the things as they really are' only by abstracting from the means of observation. As long as we explore our world's most general, crude, and macroscopic featurs, we can always find such means of observation and exploration which do not appreciably change the picture, stay out of it for all practical purposes. Then the abstraction from the means of observation is valid. The light which we need to see

apples does not knock them down from the tree. But at some point in the refinement of scientific knowledge we must come to the exploration, experimental and theoretical, of our means of exploration. At this point the abstraction made by the reflection theory becomes invalid, contradicting experimental facts. One of the greatest discoveries of 20th century physics is the discovery of the impossibility of the reflection theory and the necessity of the critical philosophy of nature. It became a scientifically established fact that the subject and object of knowledge are inseparable. Knowledge is not a reflection of some 'objective reality', it is a *part* of reality, one of the world's processes. While engaged in that process, we are changing the world around us.

2. Mathematical logic

Strange enough, the development of mathematics has completely ignored the revolution in the philosophy of science produced by the discoveries of physics. For the mathematicians, 'the working compromise' between empiricism and rationalism has remained in force. Mathematics was left, together with religion, in the sphere of influence of pure rationalism and Platonism. In mathematics itself, another 'working compromise' has been established. Formalism became 'the official ideology' of mathematics, while 'unofficially' the mathematicians live and work as most straightforward Platonists (see *Infinity and the Mind* by Rudy Rucker). Although many modern philosophers, for example Bertrand Russel, used the ideas and even the formalism of mathematical logic, mathematical logic itself remained based on the pre-Kantian philosophy and the reflection theory. Brouwer initiated an attack on the Platonist approach to logic and set theory from the positions of modern philosophy. Unfortunately, the intuitionist logic developed by Brouwer and his followers, as well as analogous later developments known collectively as constructivism, were unable to interpret a considerable part of "classical" mathematics, and had simply to reject it. The main body of mathematicians could not agree to such sacrifices, and Platonism-formalism prevailed.

Since its very inception, mathematical logic has been taking the reflection theory for granted. Its method and fundamental concepts were most clearly formulated by Gottlob Frege (1848-1925), who more than any other person can be considered the

author of this discipline. One of the most illuminating papers by Frege was published in 1892 under the title "Ueber Sinn and Bedeutung" (On Thought and Meaning). Frege considers *signs*, such as names, word combinations and expressions, and distinguishes the sense, or meaning, of the sign from its *nominatum*. The latter is what the sign denotes. If the lines *a*, *b* and *c* intersect at the same point, then the nominata of the expressions 'the point of intersection of *a* and *b*' and 'the point of intersection of *b* and *c*' are the same, although their meanings are different. Also, the expressions 'evening star' and 'morning star' have different senses but the same nominatum: Venus. Frege limits his consideration to those signs which function as 'proper names', i.e. have definite nominata.

Concerning the sense of signs, Frege writes: "The sense of a proper name is grasped by everyone who knows the language or the totality of designations of which the proper name is a part; this, however, illuminates the nominatum, if there is any, in a very one-sided fashion. A complete knowledge of the nominatum would require that we could tell immediately in the case of any given sense whether it belongs to the nominatum. This we shall never be able to do". So, nominata are treated by Frege as the only primary reality; the sense we put into words is needed only in order to indicate a specific nominatum, which is there anyway. The sense is always partial and often vague. Without a complete knowledge of the nominata, which of course we never have, we are unable to guarantee the establishment of a correct link between every possible sense and the corresponding nominatum. Moreover, there can be a sense which has no nominatum at all. Frege's example: "The words 'the heavenly body which has the greatest distance from the earth' have a sense; but it is very doubtful as to whether they have a nominatum". Frege concludes:"Therefore the grasping of a sense does not with certainty warrant a corresponding nominatum". He also notices that "when words are used in the customary manner then what is talked about are their nominata".

This is the conceptual basis for the predicate calculus, and it certainly is the reflection theory. For post-Kantian empiricism, it is exactly the *meaning* of words that is valuable because it relates the words to the corresponding observable facts and constitutes the only true reality. The nominata separated from the meaning are nothing more than fruits of fantasy. Yet it is exactly the meaning of words that is thrown away in the abstraction of an *object* in mathematical logic, which is its first

fundamental concept.

We proceed now to sum up the formalism of mathematical logic in its relation to the underlying philosophy and methodology. The objects of mathematical logic are supposed to exist and form a set called a *domain*. In different applications of logic, i.e. in different mathematical theories using its formalism, the domains may be different. A domain may be finite or infinite,but must not be empty. The language of mathematical logic includes *variables*: x_1, x_2 ..., etc., which can take objects from the domain as their values. An *n*-place *predicate* is denoted as a function of *n* variables: $P(x_1, \ldots, x_n)$. When some values are substituted for the variables in a predicate, the result is a *proposition*, or *sentence*. Some predicates are *primary*, that is not decomposable into more elementary units. They generate primary propositions.

If the *object* of mathematical logic is a formalization of the proper name of language, the *proposition* is a formalization of the declarative sentence. Gottlob Frege, in the paper quoted above, treats propositions from the same standpoint as he treats objects. Like a proper name, a declarative sentence has a sense and a nominatum. The sense, again, is uderstood only informally. It depends on the sense of the names used in it. If we replace one name in a sentence by another name with the same nominatum but a different sense, then the sense of the sentence will be changed. But if the sentence has a nominatum, then it certainly must not be changed by such a replacement. So, can a sentence have a nominatum? (The presumption here is, of course, that the sense of the sentence is no good for mathematical logic).

It can, answers Frege. It is its *truth value*. He writes: "By the truth-value of a sentence I mean the circumstance of its being true or false. There are no other truth-values. For brevity's sake I shall call the one the True and the other the False. Every declarative sentence, in which what matters are the nominata of the words, is therefore to be considered as a proper name; and its nominatum, if there is any, is either the True or the False. These two objects are recognized, even if only tacitly, by everyone who at all makes judgements, holds anything as true, thus even by the skeptic."

Like the objects, the propositions are stripped of all meaning when thay are admitted into the kingdom of mathematical logic. A propositions is something that can be, and *must be*,

either true or false. That is all.

This is not to say that mathematical logic has no sense or is useless. Frege's was a tremendous intellectual achievement. The very abstract nature of formal logic allows one to apply it to a wide range of phenomena, and the brilliant record of mathematical logic shows this only too clearly. But gaining in the range of a conceptual scheme, we lose in its contents. It is my impression that the failure to tackle the problem of meaning has almost always and almost universally been recognized as the fundamental weakness of formal logic.

The concept of a predicate can be better grasped using the following representation of it, which we are tempted to accept as a sort of "interpretation", although it is not associated with any semantics or interpretation of propositions. This "interpretation" is based on the following idea. Consider a theory with the domain D. Consider some predicate $P(x_1,\ldots,x_n)$. For every n-tuple of objects from D, our predicate becomes a proposition which must be either true or false. Therefore, we are dealing with a function, which takes an n-tuple of objects from D as its argument and produces a truth-value as its value. We shall call such functions *logical*. Two predicates which for every n-tuple produce identical truth-values are essentially the same. Thus, a predicate is interpreted as a logical function of n variables from D. There are as many predicates as there are different logical functions possible.

I put quotes on "interpretation" because it is no interpretation at all. An interpretation of a theory is something that makes you closer to comparing the predictions of the theory with some facts, be they from observations of natural phenomena or from another theory. Nothing like that happens here. We simply unite a great mass of separate uninterpreted propositions into one big array, which is called 'function', but does not become less uninterpreted. It is the same as to say: 'I know who the intruders are; they are members of the intruding band'.

[The contents of the rest of Section 1.2

Logical connectives.
Quantifiers. Bound and free variables.
Two approaches to truth: deductive and quasi-semantic.

Formal systems. Axioms and rules of inference.
Quasi-semantic approach. Truth-evaluation procedure.
Universally valid formulas.
Deduction theorem.]

3. Naive set theory

[Cantor's concept of set.
Again the reflection theory in background.
Russel's paradox.
Operations on sets.
Relations and functions.
Cardinality. Uncountable sets.
The static character of math. logic and set theory.]

4. The cybernetic approach

I call my own approach to philosophy of science *cybernetic*. Essentially, it is a variety of postkantian empiricism which uses the ideas of cybernetics and our recent experience in modeling intelligent behaviour. Such an approach is as natural in the second half of the 20th century as was the use of mathematical logic by philosophers in the first half of this century. Kant started thinking in terms of how our brain processes the stream of sensations. Now, there is not less than a branch of computer science, referred to as 'Artificial Intelligence', which is engaged in modeling brain processes. Even though we are still very far from a reasonably complete understanding of the work of the human brain, we have acquired insights into this subject which should not be ignored by philosophy.

Norbert Wiener introduced the term 'cybernetics' in 1947 for the science of "control and communication in the animal and the machine". It is not my intention here to review either Wiener's ideas or their further development. I understand 'cybernetics' in a very wide sense as a science which is, so to say, complementary to physics. If physics studies the most fundamental aspects of the world which can be expressed in terms of energy-matter, cybernetics takes up the aspect of structure, organization, and information. More specifically, cybernetics is the exploration and construction of highly organized systems, by which I mean

systems with a multilevel hierarchical structure of communication and control. Computer science and technology is an important branch of cybernetics.

The philosophy of mathematics on which the theory exposed in the present book rests is based on the philosophy of science exposed in my previous book *The Phenomenon of Science: a Cybernetic Approach to Human Evolution* (Columbia University Press, 1977). In this section, I am summing up those essentials of my approach which I need here, but I realize that this summary can hardly be convincing without reading and discussing the book itself.

My approach is centered around the idea of *metasystem transition*, by which I mean a transition from a cybernetic system to a metasystem, which includes a set of systems of the initial type integrated and controlled in some manner. Each metasystem transition creates a new level in the hierarchy of control inherent in the system. Metasystem transition is *the quantum of evolution*; through the accumulation of these quanta, more and more highly organized cybernetic systems evolve.

Seen in the functional aspect, metasystem transition is the emergence of a new type of activity which can be described as *control* of the most sophisticated activity of the preceding hierarchical level. Let A be the top-level activity of a cybernetic system S, i.e. the functioning of the highest level of control in the system. For the system $S°$ formed by a metasystem transition from S, the top-level activity $A°$ can be defined by the formula:

$$A° = \text{control of}(A)$$

We can observe changes in the types of activities and conclude about metasystem transitions even without knowing the exact structure of cybernetic systems. In the evolution of life we can discern the following seven stages resulting form consecutive large-scale metasystem transitions:

1. chemical foundation of life; clusters of macromolecules positioned at random

2. control of position = **movement**

3. control of movement = **irritability** (simple reflex)

4. control of irritability = **complex reflex** (nerve net, pattern recognition)

5. control of reflex = **associating** (conditioned reflex)

6. control of associating = **human thinking**

7. control of thinking = **culture**

The processing of sensory data in higher animals is organized hierarchically. The elements of the processing machinery are *classifiers*, their states make up *representations* of the environment. The lowest-level classifiers are receptors of the sense organs; the corresponding representation is the flow of sensations. Other classifiers recognize stable complexes of sensations, thereby creating different representations and providing inputs for classifiers of the next level. They translate representations from one language to another, retaining information essential for the survival of the animal and leaving out the unessential information. Classifiers of higher levels receive information from below in the hierarchy and send it further up. Comparing this scheme with Kant's picture of the work of mind, we can loosely identify perceptions with representations created by relatively low-level space-time classifiers which are mostly common to humans and higher animals, and conceptions with the representations of higher-level classifiers.

Knowledge is the existence in a cybernetic system of a model of some part of reality as it is perceived by the system. The concept of a *model* is illustrated in Fig. 1.2. The system B models the system α if some correspondence can be established between the states of α and B such that if a state A_1 of α after some time t^a goes over into a state A_2, then the state B_1 of B corresponding to A_1 goes over, after some time t^b, into the state B_2 corresponding to A_2. To one state of B more than one state of α may correspond, while one definite state of α must correspond

to not more than one state of B.

Having a model may allow making *predictions*. The time t^b may be less than t^a. Then observing A_1, translating it into B_1 and transforming into B_2, the system can predict that a state of α will occur which corresponds to B_2, like A_2.

A cybernetic animal of stage 4 in our table (complex reflex determined hereditarily) can be said to have a knowledge of the world "implemented in hardware". If the environment of such an animal is the system α on Fig. 1.2, its instinctive responces to changes in the evironment put it in the position of the system B. Further metasystem transitions make knowledge more and more controlable, variable, transferable. On stage 5, animals become capable of learning, i.e. acquiring new models of reality during their lifetime. On stage 6, animals, who are now respectfully called humans, become capable of manipulating their images of the reality at will: the ability known as imagination. They also create languages by arbitrarily associating with the world's objects and processes certain objects of special kind: words. On stage 7, human society accumulates tremendous amounts of knowledge which is passed from generation to generation.

Language can be thought of as a continuation of the individual human brain, as the super-brain of the social super-being. Language is used to exchange knowledge between individual brains and to create new knowledge for the human society as a whole. Linguistic objects play the same role as classifiers in the brain, they implement some functional units of sensation processing which we identify as *concepts*. A concept is a symbol (a linguistic object), plus the activity in which this symbol is used. Low level concepts of natural languages simply denote certain patterns of recognition implemented in the nerve net of our brain. Other concepts are new, not existing in the brain independently of language. For example, the concept of spatial relation would be impossible if there were no *words* expressing spatial relations, like 'lower', 'to the left', 'bigger', etc. The words 'spatial relation' depend for their meaning on the existence of words which denote specific partial relations. We can see logical, i.e. implemented in language, concepts as classifiers of the social super-brain. Such classifiers as 'apple', 'lower', 'to the left', etc. make up the interface between the super-brain and the individual brain, they are based on the ability of our brain to recognize certain patterns. All the

inputs these first-level classifiers require exist independently of language. The classifier 'spatial relation' requires other logical classifiers, such as 'lower' and 'to the left' as inputs. More precisely, the activity associated with 'spatial relation' and described as understanding and using this concept, includes activities associated with other linguistic objects.

Like the individual brain, the linguistic super-brain has a hierarchical structure. Concepts which are high in the hierarchy do not allow direct translation into sense perceptions. Still they may be used in some linguistic activity which will ultimately lead to verifiable or refutable models of reality. Then they have meaning. If we see no way for linguistic activities associated with a symbol to lead outside the language to the interface with the brain and the sense organs, then this symbol has no meaning.

Linguistic activity may be either separable or inseparable from the meaning of linguistic symbols. In the former case the language is *formal*. If the system B in Fig. 1.2 is a model of reality implemented in a formal language, then transformation of the state B_1 into B_2 does not rely on human brain, and can be performed by a machine. A model created by the brain becomes independent of the brain and can be, in a metasystem transition, studied as an independent reality. This transition can be repeated again and again, and this is what we actually see in the history of exact science.

The concept of number and the science of arithmetic illustrate these ideas well. Small numbers are concepts implemented in the nerve net of our brain, they are built-in. We distinguish a collection of two objects from a collection of one or three objects as immediately as we distinguish an apple from a rabbit. The concept 'two' in human language could be seen as an "interface" concept. The concept 'one hundred and seventy' cannot be seen as an interface concept. We do not distinguish a collection of that many objects from collections referred to by neighboring numbers. Big numbers make sense only through the procedure of counting, which necessarily involves certain *linguistic objects*. The meaning of 170 includes the meaning of 169, 168, etc. The necessary linguistic objects, which we call numbers, can be of different kind, (pebbles, notches, marks of paint), but they must be there, physically, materially. They are cogwheels of the machinery of language. The process of counting is the functioning

of that machinery resulting in recognition of concepts like 17Ø, much as the functioning of certain classifiers in the brain results in recognition of the concept 'rabbit'.

Numbers, like other concepts, are used to model reality and make predictions. The statement 5+7=12 is a model with which we predict that if we have a collection of objects which was recognized as 5 by the process of counting, and another collection, which was recognized as 7, then merging these two will produce a collection which will be recognized as 12. This model is applied in the non-linguistic world: to such objects as apples, rabbits, or whatever we care to count. Numbers, and operations on them, can be completely formalized; then we deal with material objects and processes as definite and objective as celestial bodies and their movements. We call them linguistic objects, linguistic processes, and linguistic machines.

If specific numbers are first-level linguistic concepts applied directly to the non-linguistic world, such concepts as 'number' or 'arithmetic operation' belong to the next level of the hierarchy. They are not applied -- directly -- to apples and rabbits; we use them to construct models of linguistic processes. Example of such a model: the sum of two numbers does not depend on the order of the items. As we introduce the linguistic object 5 to correspond to five apples, five rabbits, etc., we introduce the linguistic object *x* to correspond to 5, 7, etc., which gives rise to algebra. Having created primary linguistic models of world's processes, we create models of these models, then models of models of models, etc. It is by metasystem transitions of this kind that the infrastructure of mathematics has been created.

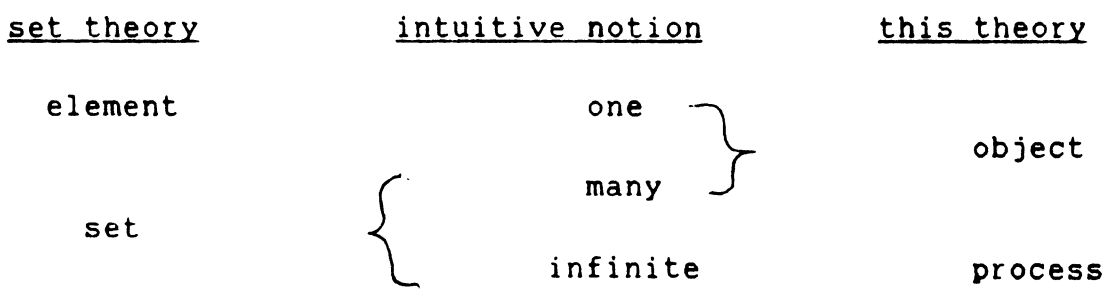
Science creates linguistic models of reality. When we deal with concepts and models of the first level, which relate directly to observable facts, we are engaged in natural sciences. Equations of arithmetic, such as 5+7=12, belong to natural science to no less an extent than Newton's equations. So do propositions expressing properties of geometric figures. When we deal with concepts and models which pertain to a special kind of reality, namely formal linguistic objects and machines, we work in mathematics proper. *Proving* that geometric figures have certain properties belongs to mathematics, as well as proving theorems about any formal objects, of course.

One can see from the preceding that our approach to the

foundations of mathematics is inherently dynamic, that is the idea of time is one of its explicite conceptual constituents. This stands in a sharp contrast with set theory, which is inherently static. The difference stems from the different philosophical backgrounds of the two approaches. Set theory is based on the reflection theory, according to which our thought is a reflection of reality. Both reality and thought are seen as instantaneous still pictures. Time is allowed here to change reality, but it is not allowed *into* reality, nor into the relation between reality and language. Actual infinity of set theory is an attempt to represent the concept of potential infinity (which includes the idea of time) in a static theory.

Our theory is based on the cybernetic view of the human being according to which our thought is the processing of our sensations. The idea of time enters from the very beginning through the concepts of a process and a model. The concept of process is our first ontological primitive.

Processes may be infinite, and this is he only infinity we allow in our theory. In fact, a process is always infinite, because this concept is a formalization of our intuitive notion of time, and we find no end of time in our intuition. What we call a 'finite' process is *a finite search*, that is such a process which at some time achieves a certain stage, after which we are not interested in the process any more. Objects we consider are always finite, although this does not mean that we put some restraints on their size. Any collection, or set, of objects which we treat as an object in our theory must be finite. The following diagram summarizes the formalization of the intuitive notions of *one*, *many*, and *infinite* in set theory and in our theory:



C H A P T E R 2

Objects, Processes, Machines

1. The Turing machine

An important step towards putting mathematics on the empiricist track was made by Alan M. Turing. In 1936, he introduced in mathematics an abstract device, or rather a class of devices, which became known as *Turing machines*. The idea was to make *mathematical computations* an object of mathematical study. Computation is understood in the widest sense: arithmetic calculations, manipulation of algebraic expressions, formal logical deductions, etc. The common feature of these operations is that they are conducted according to some rules, which the mathematician can formulate in such a precise and exhaustive manner that the operations could be performed by a machine. Turing wanted a type of machine so general that any conceivable computation could be described as the work of one of the machines of that type.

[The definition and examples of Turing machines.]

2. Refal, informally

The Refal machine will be formally defined in the next section. In this section we discuss informally, by way of introduction, the main features of Refal, so that the reader can easier see the origin and the purpose of every article of the formal definition.

'Refal' is the acronym for **RE***cursive* **F***unctions* **A***lgorithmic* **L***anguage*, a computer programming language which was developed by the author and co-workers in 1966-1970 (see bibliography in *Turchin, 1980*) and is implemented on several computer systems, including the IBM/370. Refal as an algorithmic language is con-

ceived to be simple enough to allow mathematical treatment but still successful as a practical programming language for such fields as artificial intelligence and word processing. A guide for programming in Refal can be found in *Turchin, 1980*.

We shall discuss three types of applications, or purposes of Refal, in the order of increasing generality.

Refal can be seen as a language of semantic descriptions. Consider a linguistic object which has some meaning. What does it mean to understand its meaning? It is to know how to relate it to some primary reality. Well-developed languages usually include a hierarchy of concepts, so that to relate a linguistic object to reality you have to go through a sequence of *concretizations*, that is steps which express the meaning of this object in terms of concepts that take lower positions in the hierarchy. With a natural language, the process of concretization comes to an end when relations are established between the linguistic object and the world of sensations. In the case of a formal language, concretizations lead to the concepts defined as the primitives.

The following line:

(1) <ACM> → Association for Computing Machinery

is a *sentence* of Refal. The angular brackets are *concretization brackets*. They enclose a linguistic object which must be concretized, that is replaced by linguistic objects which in some sense are closer to the ultimate reality. The sentence (1) expresses the expansion of the acronym ACM. It consists of the left side and the right side separated by the arrow →. The Refal machine, i.e. the device that "understands" Refal, takes this sentence as the instruction to replace <ACM> by Association for Computing Machinery. The letters A,C,M,A,s,s,o,... etc. will be referred to as *symbols*. Angular brackets are *special signs* of Refal, not symbols.

Consider another sentence:

(2) <the first symbol of $s_1 e_2$> → s_1

It defines what the first symbol of an expression is, and can be translated as: the first symbol of an expression which consists of a symbol s_1 after which an expression e_2 immediately follows, is s_1. Here s_1 and e_2 are free variables. The first is a symbol

variable (s-variable, for short), the second an expression variable (e-variable). This sentence is used in the following way. Suppose we observe the expression

(3) <the first symbol of APPLE>

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

Now let us look at Refal from another angle, namely as the language of recursive functions. Let us replace the string 'the first symbol of' by the single symbol F:

(4) $\langle F\ s_1 e_2 \rangle \rightarrow s_1$

We can see the replacemnt of <F APPLE> by A as the evaluation of a function call. Then (4) defines the function F whose value is the first symbol of its argument. The angular brackets should be called *evaluation brackets*. They enclose the expressions which must be understood as function calls and evaluated. <F *A*> corresponds to F(*A*) in the usual notation. Besides angular brackets, which indicate evaluation, we use in Refal usual round brackets (parentheses). They serve a different purpose: to give a structure to expressions. Any sequence of symbols and parentheses in which the parentheses are properly paired is a legitimate expression in Refal. Parentheses, like concretization/evaluation brackets, are not symbols, but special signs. Here are examples of Refal expressions:

```
A
ABC
A+B()
(bbb+(**))(())+-
```

An empty expression (just nothing) is also a legitimate expres-

sion. The argument of a Refal function can always be considered as one expression. If we want to define a function of several arguments, we use parentheses to combine them into one expression so that when necessary, it could be uniquely broken down into the original constituents. For instance, the function which concatenates its two arguments can be defined as

$$\langle \underline{\text{conc}}\ (e_x)(e_y)\rangle \ \rightarrow\ e_x e_y$$

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

Consider the following group of sentences:

(5.1) $\langle \underline{\text{chpm}} + e_x\rangle \ \rightarrow\ -\langle \underline{\text{chpm}}\ e_x\rangle$
(5.2) $\langle \underline{\text{chpm}}\ s_a e_x\rangle \ \rightarrow\ s_a\ \langle \underline{\text{chpm}}\ e_x\rangle$
(5.3) $\langle \underline{\text{chpm}}\ (e_a)e_x\rangle \ \rightarrow\ (e_a)\langle\ \underline{\text{chpm}}\ e_x\rangle$
(5.4) $\langle \underline{\text{chpm}}\rangle \ \rightarrow$

It defines the function chpm, 'change plus to minus'. If this function is applied to an expression, its value will be the result of the replacement of every sign + on the top level of the bracket structure in the argument by the sign - Consider this function call

(6.1) $\langle \underline{\text{chpm}}\ c+(a+bx)\rangle$

The Refal machine will evaluate this call by steps, each step being the application of one sentence. It will try to apply sentences *in the order they are listed*. When a sentence is found applicable, it is applied, and this is the end of the step; on the next evaluation step, the Refal machine will try to apply sentences starting with the first one again. The expression to be evaluated is said to be in the *view-field* of the Refal machine. The expression (6.1) in our example is the initial state of the view-field.

Making the first step, the Refal machine tries to apply the sentence (5.1), but of course fails, because the argumant does not start with +. Then it tries to apply (5.2) and this time succeeds. The view-field becomes:

(6.2) $C\langle \underline{\text{chpm}}\ +(A+BX)\rangle$

The symbol C is in the view-field outside of any evaluation brackets. It means that it will never be changed into anything else, and will be there till the end of the work. The Refal machine alters only those expressions which are bounded by evaluation brackets.

The results of the consecutive steps after (6.2) are as follows:

(6.3)	C-<chpm (A+BX)>	by (5.1)
(6.4)	C-(A+BX)<chpm>	by (5.3)
(6.5)	C-(A+BX)	by(5.4)

In the state (6.5) of the view-field there are no evaluation brackets; the Refal machine comes to a normal stop, and the contents of the view-field is the result of evaluation. Note that in order to compute the initial call (6.1), the function chpm called itself (with a different argument) in (6.2). The sentences (5.1), (5.2) and (5.3) define the value of a function call through the call of the same function. Functions defined in this way are called *ecursive*.

Now we are going to discuss the third, and the most general view of the language Refal and the Refal machine. The Refal machine is a framework for *the linguistic representation of the world*.

We see the world as the interplay of various *processes*, which involve various *objects*. We can change objects ourselves, thereby giving rise to processes. We also can create and start *machines*, which maintain processes autonomously. The concepts of object, process, and machine will be considered primary and given to us intuitively. We can only define them informally for clarification, and characterize their relationship.

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

Among all the objects we deal with we choose some to serve the purposes of communication and cognition. This is the phenomenon of *language*. The objects so used are referred to as *linguistic objects*. In this book we confine ourselves to one-dimensional discrete languages, in which linguistic objects are strings of distinct *characters*. A process the stages of which are linguistic objects is a *linguistic process*. A machine which gives rise to a linguistic process and takes linguistic objects as its input is a *linguistic machine*. The Refal machine is a linguistic machine which can also be called a *metamachine*, because it is used not only to define linguistic processes, but also linguistic machines.

Semantic definitions in the theory of languages and computations in mathematics emerged and acquired significance because they are part of a system of linguistic representation of the world's processes. Thus, linguistic representation of the world and the managing of the processes in this representation is the most general kind of activity we are engaged in as scientists. Refal will be used here as the language of this representation.

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

3. Formal definition of Refal

We present here a formal definition of the version of the language known as *strict Refal*.

The 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; a *specifier* (see below) may appear between the letter s and its subscript, e.g. $s(ABC)_1$, $s(+-)_\alpha$.

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^1,
• strings of characters underlined to form one (composite) symbol, e.g., then.

We shall 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. can not be represented as

$$E_1\ e_1\ E_2\ e_J\ E_3\ ,$$

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

$$Ae_1,\ BC(DE),\ e_1+(e_2)(e_3),\ s_1e_xs_1$$

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:

$$\langle L\rangle = R$$

where L is a pattern expression and R is an arbitrary (general) expression of Refal. The equality sign 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 $E_1, E_2, \ldots, E_n$ is the expression

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

• 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_I, where I is any index, can take as its value any symbol.
(b) A specified s-variable $s(P)_I$, where P is a string of symbols, can take as its value any of the symbols entering P; string P is called a *specifier*.
(c) An e-variable e_I can take any expression as its value.
(d) 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 substitutiopn 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 <?>, 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 for the future to note that an indefinite process is infinite.

4. Examples of processes and machines

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

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

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

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

Now let us add to the program the following two sentences:

```
<Be_x*> → <Be_x>
<B> → end
```

If we put <B***> into the view-field, the consecutive stages of the process will be:

```
<B***>
<B**>
<B*>
<B>
end
```

This is a finite process with the object symbol end as its final stage.

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

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

For instance, with the above sentences , the format $\langle B e_x \rangle$ defines a machine which for every string of asterisks as its input e_x generates a finite process. If we substitute a different kind of value for e_x the resulting process will be undefined. The machine $\langle \alpha e_x \rangle$ generates an infinite process for every input expression substituted for e_x (it infinitely adds characters A on the right).

Let us consider less trivial examples. In the unary number system, where zero is represented by Ø, one by Ø1, two by Ø11,

etc., the adding machine with the format

$$\langle +(e_x)e_y \rangle$$

can be defined by the program

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

With the input values $\emptyset 1$ for e_x and $\emptyset 11$ for e_y this machine will generate a finite computation process:

$\langle +(\emptyset 1)\emptyset 11 \rangle$
$\langle +(\emptyset 1)\emptyset 1 \rangle$ 1
$\langle +(\emptyset 1)\emptyset \rangle$ 11
$\emptyset 111$

We could define an equivalent machine choosing a different format, e.g., $\langle +(e_x)(e_y) \rangle$, or $\langle \underline{\text{add}} e_x, e_y \rangle$, etc.

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

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

The format is $\langle \underline{\text{add}}(e_1)e_2 \rangle$. (Note that the variables we choose to represent formats are not related in any way to the variables used in programs; neither are variables in different sentences of the program. But we usually keep to the same variables as a matter of convenience). The last sentence of the program for add may not be understood immediately. It will work correctly because it will be used only in the situation when at least on of the two arguments e_x and e_y is empty. The program would be more understandable if instead of that sentence we used these two:

$$\langle \underline{\text{add}}(e_x) \rangle \rightarrow e_x$$
$$\langle \underline{\text{add}}()e_y \rangle \rightarrow e_y$$

Exercise. Trace the process $\langle \underline{\text{add}}(1\emptyset\emptyset 1\emptyset)1\emptyset 1 \rangle$ using the formal definition of the Refal machine in all its detail.

5. Metacode and Self-Simulation

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

In the Refal machine	In the metacode
s_I	*SI
$s(P)_I$	*S(P)I
e_I	*EI
<	*(
>	)
*	*V
S	S

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

A program consisting of sentences $z_1, z_2, \ldots, z_n$ will become

$$(\uparrow z_1)(\uparrow z_2) \ldots (\uparrow z_n)$$

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

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

We shall also use a *shorthand notation* for metacoded vari-

ables, by which E_x^1 stands for *EX, E_x^n stands for $\uparrow E_x^{n-1}$, and the analogous convention holds for s-variables. The superscript ^{1} can be omitted. Examples follow:

Shorthand	Strict notation	meaning
E_1	*E1	$\uparrow e_1$
E_a^2	*VEA	$\uparrow\uparrow e_a$
S_x	*SX	$\uparrow e_x$
$S^4(?!)_3$	*VVVS(?!)3	$\uparrow\uparrow\uparrow\uparrow s(?!)_3$

The asterisk * is singled out to represent special Refal signs in metacode, so it must undergo change in the metacode transformation for the inverse transformation to be unique. An object expression which may contain asterisks can be transformed into metacode by using the function $\langle \mu e_x \rangle$ defined by:

$$\langle \mu\ *e_1 \rangle \rightarrow *V\ \langle \mu\ e_1 \rangle$$
$$\langle \mu\ s_a e_1 \rangle \rightarrow s_a\ \langle \mu e_1 \rangle$$
$$\langle \mu\ (e_1)e_2 \rangle \rightarrow (\langle \mu\ e_1 \rangle)\ \langle \mu\ e_2 \rangle$$
$$\langle \mu\ \rangle \rightarrow$$

The inverse function $\langle \bar{\mu}\ e_x \rangle$ is defined correspondingly:

$$\langle \bar{\mu} *V e_1 \rangle \rightarrow *\ \langle \bar{\mu} e_1 \rangle$$
$$\langle \bar{\mu} s_a e_1 \rangle \rightarrow s_a\ \langle \bar{\mu} e_1 \rangle$$
$$\langle \bar{\mu} (e_1)e_2 \rangle \rightarrow (\langle \bar{\mu} e_1 \rangle)\ \langle \bar{\mu} e_2 \rangle$$
$$\langle \bar{\mu} \rangle \rightarrow$$

We introduce now a machine $\langle \underline{\text{stepu}}(e_p)e_a \rangle$ ("step-universal"), which simulates one step of the Refal machine. If *P* is a process loaded into the program field of the Refal machine, and *A* is a process expression put into its view-field, then the process $\langle \underline{\text{stepu}}(\uparrow X)\uparrow A \rangle$ is always finite and its result is the metacode of the expression to be found in the view-field after the Refal machine has made exactly one step. Using the <u>stepu</u> machine we can define the <u>actu</u> machine ("activate-universal"), which simulates fully the operation of the Refal machine, as if activating the metacode representation $\uparrow A$ of a process *A* into the process itself. Using repeatedly the <u>stepu</u> machine, the <u>actu</u> machine passes over from the current state of the view-field to the next. If a

Refal machine loaded as above generates a finite process then the process ⟨actu(↑*P*)↑*A*⟩ is also finite and produces the same result but in the metacode; otherwise it is infinite. Machines stepu and actu can be defined in Refal; we shall not list their definitions, though.

Whenever we run the Refal machine, there is a certain program *P* in its memory (program field). The machines we actually need and shall use in the following simulate the Refal machine loaded with a specific program *P*, namely the one that includes all the definitions we have done up to date, and only them. These machines are step and act:

⟨step e_x⟩ → ⟨stepu(↑*P*)e_x⟩
⟨act e_x⟩ → ⟨actu(↑*P*)e_x⟩

Thus ⟨step ↑*A*⟩ will produce the metacode of the next stage of the view-field after *A* under the current cumulative program *P*, and ⟨act↑*A*⟩ will activate *A* under the same program.

6. Searches and generators

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

An expression is *passive* if it includes no activation brackets. A *search* is a process each stage of which is either of the form ⟨*E*⟩ or passive. The latter case takes place, obviously, at the end of a finite search. The terminal stage of a search will be referred to as its *result*. An infinite search produces no result. Processes ⟨α*A*⟩ and ⟨β***⟩ with α and β defined as above are searches, although these are not very edifying examples. A search, as the name suggests, is a process which you would typically initiate in order to find (construct) a certain object: the result of the search.

A *generator* is a process each stage of which is either *L*⟨*E*⟩ or *L*, where *L* is a list of object expressions. Recall that a *list* is an expression of the form

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

where *n* can be any number, including zero (an empty list). The

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

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

The process <num Ø> is a generator of all natural numbers. The process <num *N*> generates the set of all numbers that are greater then, or equal to *N*.

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

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

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

$$\langle +(e_1)Ø \rangle \rightarrow e_1$$
$$\langle +(e_1)e_2 1 \rangle \rightarrow \langle \underline{\text{out}}\langle +(e_1)e_2 \rangle 1 \rangle$$

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

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

```
<out 0111>
0111
```

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

$$L \rightarrow R$$

we allow the following two sentential forms:

(s) $L \rightarrow s \mid R^1 \mid R^2$

and

(g) $L \rightarrow g \mid R^1 \mid R^2$

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

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

generator which used sentence (g).

The Refal machine endowed with the described abilities may be called "parallelistic". It can execute searches and set generations in parallel. Since each of the branches R^1 and R^2 can again use a sentence of the form (s) or (g) the parallelistic Refal machine can generate a potentially infinite tree of parallel branches.

As mentioned before, the parallelistic Refal machine can be easily simulated on the regular (sequential) Refal machine. For that, every sentence of the form (s) must be replaced by

(s') $L \rightarrow \langle \underline{pars}(\uparrow R^1)(\uparrow R^2)\rangle$

and every sentence of the form (g) must be replaced by

(g') $L \rightarrow \langle \underline{parg}(\uparrow R^1)(\uparrow R^2)\rangle$

The machines pars (parallel searches) and parg (parallel generators) are defined as follows:

$\langle \underline{pars}(*(e_1))(e_2) \rightarrow \langle \underline{pars}(e_2)(\langle \underline{step}*(e_1)\rangle)\rangle$
$\langle \underline{pars}(e_p)(e_2)\rangle \rightarrow \langle \bar{\mu} e_p\rangle$

$\langle \underline{parg}((e_m)e_1)(e_2) \rightarrow (\langle \bar{\mu} e_m\rangle)\ \langle \underline{parg}(e_1)(e_2)\rangle$
$\langle \underline{parg}(*(e_1))(e_2)\rangle \rightarrow \langle \underline{parg}(e_2)(\langle \underline{step}*(e_1)\rangle)\rangle$
$\langle \underline{parg}()(e_2)\rangle \rightarrow \langle \underline{act}\ e_2\rangle$

We can think of all our models as taking place in the sequential Refal machine and see (s) and (g) as a convenient representation of (s') and (g').

Propositons

1. Models,Selections,Predictions

We proceed now to examine the intuitive notion of a *model*. Acccording to the philosophy outlined in Chapter 1, a mathematical proposition has a meaning to the extent it produces some models of reality. Now we want to formalize the notion of a model and find something like minimal units of semantics, some elementary propositions, combining which we could construct every meaningful proposition.

Informally, we say that the process *B models* the process *A* if there is some relation or similarity between the stages of *B* and *A*. It is not necessary that every stage of *A* or *B* be related to some stage of the other process; generally, we select some stages in *B* which should be somehow put into correspondence with some stages selected from *A*. The notion of a relation or a correspondence between the stages of two processes will be discussed and formalized a bit later. First, let us deal with the general structure of a model.

Marking some of the stages of a process as selected for a certain purpose creates what we shall call a *selection*. In Fig.3.1, the stages of process *B* at moments of time *t= 2,3,6,8*, etc. make up a selection; so do stages *t=1,4,6,11*, etc. of *A*. The selected stages will be called *the members* of the selection; the process whose stages are selected will be referred to as the *underlying* process.

The concept of a selection is a better approximation to our perception of the world than the concept of a discrete process. Non-linguistic processes may be continuous; representing them as discrete with a certain choice of time interval includes a good deal of arbitrariness. But speaking of a selection we must not necessarily specify how many stages of the underlying process are left between its neighboring members. The criterion for selecting a stage can use the current configuration of the process and not the sequential number of that stage in a discrete representation

of the process. Suppose, for instance, that we write down the readings of a measuring instrument at certain moments of time. This is a selection, and it is the same no matter what time interval is chosen to represent the underlying process as discrete.

Unlike processes, which are essentially infinite because time never stops, selections may be genuinely finite. If after a certain stage of a process all the subsequent stages do not satisfy the criterion of a selection, there will be no more members of this selection: it is finite. The notion of finiteness when applied to a selection is very different from when it is applied to a search. The finiteness of a search can be directly verified, while the finiteness of a selection cannot. Verifying that a search is finite, we discover a stage which satisfies the search (is selected), and do not care what happens to the process afterwards. To establish that a selection is finite, we have to prove that after a certain stage, no selected stage will ever appear. Speaking about finite searches and processes defined in Refal, we identify the end with the appearance of a passive stage. From the definition of the Refal machine it follows that once a passive stage appears in the view-field, it will never change. But we cannot verify this directly, of course. So, the finiteness of a Refal process can be seen both as the finiteness of a search for a passive stage, and as the finiteness of a selection with active stages selected. In the first case it is directly verifiable; in the second case we must add a little bit of belief.

A selection can be seen as a sequence of searches, which lead to the consecutive selected stages of the underlying process. One of them may be infinite, in which case it is the last one. A selection is finite if and only if the number of constituent searches is finite and the last search is infinite.

The assertion that the process *B* in Fig.3.1 models the process *A* means that on each of the processes a selection is defined and the members of these selections with the same sequential numbers are in a certain relation, which is shown by dashes in Fig.3.1. (It is not necessary, though, that the relation is the same for all pairs). This assertion can be split into as many constituent parts as there are pairs of selected stages.

Consider the first part of this assertion, namely that the first members of the two selections are in a certain relation. We

shall call such an assertion a *prediction*. In a typical case, one of the processes, say A, will be the subject of the assertion, while the other, i.e. B, will be created (defined) to formulate the assertion (to serve as a model of A). Process B is always linguistic, by the definition of language. Process A may be either natural, like planet movement, chemical reaction in blood, etc., or linguistic, like multiplication of numbers; in the former case the assertion belongs to the natural sciences, in the latter -- to mathematics.

The meaning of an assertion is the way we use it as a model of reality. It is as follows. We first activate linguistic process B and run it until the first selected stage is reached; in other words we run the first constituent search of the selection based on B. Let us denote it B^1. This would usually be referred to as a computation or logical derivation, or whatever. The search B^1 comes, presumably, to an end, producing a certain resulting expression R_b. Now we predict (which means that we take it as the basis for making decisions) that the process A will come to a selected stage, i.e. the first constituent search A^1 will stop, with a result R_a which will stay in a certain relation to R_b (e.g., will be measured by the number B). This is essentially the same definition of a predicting model as given in Chapter 1 and illustrated in Fig.1.2.

We must now formalize the notion of a relation between objects. It is here where the difference between natural sciences and mathematics manifests itself. In natural sciences the subjects of proposition, the process A, is non-linguistic. To define how its stages, non-linguistic objects, are related to linguistic objects of B, we must necessarily use procedures which are not purely linguistic. They can be referred to as procedures of *measurement*. Using this term we generalize the usual concept of measurement which results in a number. It will now denote any procedure which can be applied to a non-linguistic object and produce a lingusitic object of a theory.

In mathematics, the process A is linguistic; therefore the relation between its stages and those of the model B which makes a proposition meaningful can be defined by a linguistic machine. Consequently, the semantics of mathematics can be completely formalized within the linguistic sub-universe.

We assume that a relation ρ between linguistic objects is defined if a *testing device* $\langle \rho(e_x)e_y \rangle$ is constructed, which for every pair e_x, e_y generates a process (a *search*, to be more exact), which is finite if and only if e_x and e_y are in relation ρ. By a *device* we mean anything that helps to originate a process when given a number of objects as input. A device is essentially a parameterized process, and we will often use the term process when referring to an expression with free variables, instead of using the term device. If we cannot define such a device then we do not know what we are speaking about; this is our fundamental philosophical principle.

There are a lot of propositions in mathematics which express relations between linguistic objects and are thought of as exactly and formally defined by mathematicians, but do not qualify according to our definition if a testing device is understood to be a Turing machine or its equivalent. (As an example, take a set of numbers *S* which is not recursively enumerable, and consider the property of a number *n* to be an element of *S*: *n* el *S*. There exists no Turing machine which stops if and only if *n* el *S*). Our definition of a device, however, is less restrictive. Our theory allows reference to special *real-time* processes, which cannot be modeled on a Turing machine. Their nature and role in our theory will be discussed later. The general principle that the meaning of every proposition must be expressed in terms of real processes, and not by invoking the idea of actual infinity, remains universally valid.

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

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

which stops if and only if σ produces T. Analogously we define a

testing machine ρ_f which stops if and only if σ produces F. Running the two testing machines in parallel will allow us to do anything that can be done by running σ. The converse statement, that whatever can be done with semi-predicates can also be done with total recursive predicates, would not be true, of course. Mathematics knows a lot of relations which can be defined by a semi-predicate, but not by a total recursive predicate (non-decidable but recursively enumerable sets). The semi-predicate is a smaller semantic unit than the recursive predicate: "one half" of it.

Return to the prediction concept. To characterize a prediction we must define two searches: $\langle F^b \rangle$ and $\langle F^a \rangle$, and one relation $\langle \rho(e_x)e_y \rangle$. However, it is a certain combined usage of the three that constitutes the meaning of the prediction. The prediction we are speaking of is the assertion that the process

$$C = \langle \rho(\langle F^b \uparrow B \rangle)\langle F^a \uparrow A \rangle \rangle$$

is finite. Indeed, according to the definition of the Refal machine, when we put C into the view-field the first subexpression to become active will be $\langle F^b \uparrow B \rangle$. This process will be initiated, and when it stops (if it does) the other search, $\langle F^a \uparrow A \rangle$, will be initiated. When and if it comes to an end, the ρ process will be run over the results of the two former processes. The prediction as we defined it above asserts that there is a selected stage to occur in process A and in process B, and that these stages are in the relation ρ. This is equivalent to the statement that process C is finite.

The statement of the finiteness of the process C still fits our general definition of prediction, but it is a prediction of a special kind. Here the process being modeled (grammatical subject) is C; the modeling process (grammatical attribute) is any process known for sure to be finite, e.g., simply the empty expression []; and the testing machine (grammatical 'is') always says yes no matter what the arguments are, e.g. as defined by:

$$\langle \rho(e_x)e_y \rangle \rightarrow$$

As we have seen, every prediction can be represented as a prediction of this special kind: that a given process (search) is finite. This will be our elementary semantic unit, the quantum of semantics. When speaking of predictions further on we shall refer

to these special predictions. We introduce a notation for them. The statement that a process *A* is finite will be represented by the object expression

$$\uparrow A!$$

which is read: "*A* is finite".

Examples. The statement that a process <αAAA> is finite is the prediction *(αAAA)!. If the α-machine is defined as in Chapter 2 by

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

then this is a false prediction, because this process is infinite. With the machine B defined as in Chapter 2, the process <B***> is finite. The corresponding prediction:

$$*(B*V*V*V)!$$

is true. Note that the asterisk * turns into *V in the metacode. Because of this it is not advisable to use the asterisk in any context where another symbol can be used instead. The use of asterisk should be reserved for the representation of free variables and activation brackets in metacode, where the asterisk's unique feature plays a useful role: it provides for the uniqueness of the inverse transformation.

2. Propositions

A process *A'* which starts with some, but not the first, stage of a process *A* is a *descendant* of *A*. When we use a model, like the one in Fig.3.1, we first obtain a prediction from it, as described above. After the completion of the searches based on the processes *B* and *A* we have a descendant of *B* (which in Fig.3.1 starts with the stage $t=2$) and a descendant of *A* (which starts with $t=1$). These descendants are again in the relation of modelling, which gives us a new prediction, and so on. We see that a model is, essentially, *a generator of predictions*.

If a model generates only a finite set of predictions, it can be represented by a list of these predictions. In the general case, when the set of predictions may be infinite, we have to

represent the model by the generator itself. To become an object expression the generator must be metacoded. If the simplest form of a meaningful mathematical proposition is a prediction, a more general kind is a generator of predictions. Recall that we represent generators by processes of special kind defined in Refal. If *G* is a Refal process generating only predictions, the object expression ↑*G* represents a proposition whose meaning, intuitively, is the set of all predictions produced by *G*. Syntactically, we easily distinguish prediction-propositions from generator-propositions because the former always end with '!', while the latter, if not empty, always end with a right parenthesis. The ability to distinguish between object expressions representing predictions and those representing generators is, of course, absolutely essential, because their use (and therefore meaning) is different. Predictions are, so to say, ready for use. To use a generator we must run it and use the predictions it produces.

Our first example of a generator proposition is the statement that a given process is infinite. What is its meaning? Can it be understood as a generator of predictions?

Yes, to state that a process *A* is infinite is to state that:
(1) the initial stage *A* is not passive (includes at least one pair of activation brackets);
(2) the next stage after the initial stage is not passive;
(3) the next stage after the next stage after the initial stage is not passive;
(4) the next stage after the next stage after the next stage after the initial ...
and so on infinitely. Every one of these statements can be formalized as a prediction by defining a process which checks whether a given process expression is not passive and stating that the process when applied to a given stage of *A* is finite. Thus the infinity of a certain process is an infinite generator of predictions.

The infinity model for a process *A* is shown in Fig.3.2. The attribute process *B* is a trivial process each stage of which is the empty expression. The relation ρ is the property <u>npas</u> of a process stage to be not passive. The necessary definitions are:

#R $\langle \rho()e_x \rangle \rightarrow \langle \underline{npas}\ e_x \rangle$

#1 $\langle \underline{npas}\ *(e_x)e_y \rangle \rightarrow$
#2 $\langle \underline{npas}\ s_i e_x \rangle \rightarrow \langle \underline{npas}\ e_x \rangle$
#3 $\langle \underline{npas}\ (e_x)e_y \rangle \rightarrow \langle \underline{npas}\ e_x\ e_y \rangle$
#4 $\langle \underline{npas}\ e_x \rangle \rightarrow \langle \underline{npas}\ e_x \rangle$

Let us see how these definitions work. The function ρ simply discards the first argument (which is always empty) and calls function npas, 'non-passive'. Sentence #1 is applied when the argument starts with an activation bracket, which in the metacode is represented by the asterisk and a left parenthesis. In this case the npas machine stops, because the argument is non-passive. Sentences #2 and #3 define the process of scanning the argument from left to right until the combination recognized by #1 is met (if it is). Sentence #2 is applied when either the first symbol in the argument is not asterisk, or it is an asterisk but is not followed by a left parenthesis. This symbol is eliminated and function npas is applied to the remainder. Sentence #3 eliminates those parentheses (structure brackets) which in the metacode represent themselves, and not activation brackets. If the whole argument gets destroyed by #2 and #3 without finding an activation bracket, then sentence #4 will be applied, which results in an infinite process ⟨npas⟩. Thus ⟨npas *E*⟩ is finite if and only if the expression *E* represents in the metacode a non-passive expression.

To create generators representing infinity models such as in Fig.3.2, we want a machine which for any given argument ↑*A*, where *A* is a search, will generate the infinite set of predictions as discussed above. Let us see what these predictions should be. The initial stage is *A*. The process checking that *A* is non-passive is ⟨npas ↑*A*⟩. The prediction that this process is finite is its metacode followed by '!', i.e.

*(npas ↑↑*A*)!

If we denote by *A'* the stage immediately following *A*, its metacode ↑*A'* can be computed as ⟨step ↑*A*⟩, its double metacode ↑↑*A'* is *(step ↑↑*A*). The prediction that *A'* is passive is obtained from the above expression by substituting ↑↑*A'* for ↑↑*A*. It is:

*(npas *(step ↑↑*A*))!

The next prediction must be

$$*(\underline{\text{npas}}\ *(\underline{\text{step}}\ *(\underline{\text{step}}\ \uparrow\uparrow A)))!$$

etc.

We give the name <u>inf</u> to the machine we want. It can be defined as follows:

$$\langle \underline{\text{inf}}\ e_p \rangle \rightarrow (*(\underline{\text{npas}}\ e_p)!)\ \langle \underline{\text{inf}}\ *(\underline{\text{step}}\ e_p) \rangle$$

Now the proposition that a given process *A* is infinite is the metacode of the generator $\langle \underline{\text{inf}}\ \uparrow\uparrow A \rangle$, i.e. $*(\underline{\text{inf}}\ \uparrow\uparrow\uparrow A)$.

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

Example. The statement that the process $\langle \alpha AAA \rangle$ is infinite is $*(\alpha AAA)?$, or $*(\underline{\text{inf}}\ *V(\alpha AAA))$.

We came close to a general definition of proposition. We have only to make one last generalization. It is not necessary that a generator produce only predictions ready for use. It may also produce generators, which in their turn produce predictions and generators, which produce predictions and ... etc. We come to the following inductive definition:

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

Thus a proposition may produce a whole hierarchy of propositions, but they must be such that *ultimately* they produce *predictions*. A formal object has a meaning as a proposition only to the extent we know how to make it produce predictions. If there is no way to obtain predictions from an object, it has no meaning as a proposition. Atomic propositions constitute the ground level of

the hierarchy of propositions. We recognize them syntactically by the fact that they end with a symbol '!' or '?'. If a proposition does not end with one of these it should be treated as the metacode of a machine which is still to be run to produce lower-level propositions.

Examples. Consider the following proposition of the ordinary arithmetic:

$$2+3 = 5$$

We want to represent this as a formal proposition of our theory. We shall use the unary number system and the adding machine + as defined in Section 2.4. We shall also need the tester (semi-predicate) of equality. We define it in the format $\langle=(e_1)(e_2)\rangle$ by these sentences:

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

Proposition (1) states that the process

$$\langle=(\langle+(\emptyset 11)\emptyset 111\rangle)(\emptyset 11111)\rangle$$

is finite. Hence we have

$$*(=(*(+(\emptyset 11)\emptyset 111))(\emptyset 11111))!$$

This is a prediction.

Consider the proposition:

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

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

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

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

the necessary machines will be constructed in due course.

An existentially quantified formula like

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

will be interpreted in our theory as the finiteness of the process which searches for the value of x satisfying the equation.▽

Our semantic definition of proposition leads to a natural interpretation of *logical implication*. A proposition P_1 *implies* a proposition P_2 if P_2 is among the propositions generated by P_1. This definition is the most exact formalization of the intuitive concept of logical implication, according to which if P_2 is implied by P_1, it is already somehow contained by P_1, included in it. Logicians distinguish two kinds of implication: *logical*, or *strict*, implication, and *material* implication. Material implication, unlike logical one, can connect two arbitrary propositions which in no way are related by their meaning. It establishes the connection by force, so to say, announcing it as an empirical fact, a new law of nature. Using '→' to symbolize implication, we can declare that

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

even though the definition of the concept of apple may not include that it is necessarily edible (there are inedible apples, after all). Compare this with the following implication:

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

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

In our theory we formalize both logical and material implication and one can see how different these concepts are (we shall discuss material implication in the next section). In contrast, the conventional mathematical logic has only one implication: material. The closest thing to logical implication that mathematical logic has is the concept of deducibility: Q is deducible from P if the (material) implication $P \rightarrow Q$ is a tautolo-

gy. The difference between the two kinds of implications becomes here a meta-concept referring to the way we deal with propositions, not a feature of the propositions themselves, as we understand it intuitively. This reflects, of course, the purely syntactic (formal) nature of mathematical (formal) logic, and constitutes, in our view, its main deficiency. Formal logic has nothing to do with the meaning of the constructs it introduces. For instance, when the connective and is defined, it is nowhere to be seen that (*P* and Q) *logically implies P*, so we have to state it as a *material* implication. In our theory, as we shall see later, the definition of the and connective formalizes our intuitive understanding of it, its *meaning*. Accordingly, we do not have to postulate that (*P* and Q) logically implies *P*, we prove it.

We shall define a machine which tests that one proposition logically implies another. But first we define a generator which we call int, for 'interpretation'. It takes a proposition and if it is atomic, simply gives it out. If it is non-atomic, the int machine still gives it out but then runs it as a generator and treats each emerging proposition in the same manner as the original proposition, i.e. outputs it and goes on running it if it is non-atomic. Therefore, <int *P*> produces all atomic and non-atomic propositions which can be produced by *P* directly or *hierarchically*, i.e. through the intermediate levels of the hierarchy. Naturally, such a machine must run the emerging set generators in parallel, otherwise the first infinite generator will prevent it from running other generators. The definition of *int* follows:

$$
\begin{aligned}
&\langle \underline{\text{int}}\ e_r s(!?)_i \rangle \rightarrow (e_r s_i) \\
&\langle \underline{\text{int}}\ *(e_g) \rangle \rightarrow \langle \underline{\text{int}}\ \langle \underline{\text{step}}\ *(e_g) \rangle \rangle \\
&\langle \underline{\text{int}}\ (e_r s(!?)_i) e_x \rangle \rightarrow (\langle \bar{\mu} e_r \rangle s_i) \langle \underline{\text{int}}\ e_x \rangle \\
&\qquad\qquad\qquad\qquad\qquad\quad |\langle \underline{\text{int}}\ \langle \bar{\mu} *V(e_g) \rangle \rangle \\
&\langle \underline{\text{int}}\ (*V(e_g)) e_x \rangle \rightarrow g| \\
&\qquad\qquad\qquad\qquad\qquad\quad |\langle \underline{\text{int}}\ e_x \rangle \\
&\langle \underline{\text{int}}\ \rangle \rightarrow
\end{aligned}
$$

If *P* is an atomic proposition, the int machine simply outputs this proposition. If *P* is non-atomic, then <int *P*> is a generator which produces all propositions that can be hierarchically produced starting from *P* except *P* itself. To include *P* into the set produced, we simply call <int(↑*P*)>. Indeed, (↑*P*) is a proposition which does not end with '!' or '?', so it is treated as a generator. This means that it is demetacoded and placed into

the view-field of the Refal machine. The result is (P). It is a generator which in no steps produces proposition P. Generally, if we have propositions

$$P_1, P_2, \ldots, P_n$$

we can unite them into one proposition:

$$(\uparrow P_1)(\uparrow P_2) \ldots (\uparrow P_n)$$

which is a generator producing exactly the list of the original propositions.

Compare

(2) $\quad *(\underline{int}\ (\uparrow\uparrow P))$

and

(3) $\quad \uparrow P$

The ultimate volume of propositions which can be produced starting from (2) and from (3) is the same, but (2) produces them all at once, while (3) if simply run produces only the top level of the hierarchy.

Using the proposition interpretation machine <u>int</u> we define the implication machine <u>imp</u> such that the process

$$\langle \underline{imp}\ (P)\rightarrow Q\rangle$$

tests whether (i.e. end if and only if) Q is among the propositions that can be hierarchically produced starting from P. (The symbol '→' in the format of <u>imp</u> is just for readability). The definition of <u>imp</u> is:

$$\langle \underline{imp}(e_p)\rightarrow e_q\rangle \rightarrow \langle \underline{elm}(\langle \mu e_q\rangle)\underline{of}\ *(\underline{int}(\langle \mu\langle \mu e_p\rangle\rangle))\rangle$$

It uses an auxiliary function <u>elm</u>, 'element-metacode', such that $\langle \underline{elm}(e_p)\underline{of}\ e_g\rangle$ tests whether e_p is the metacode of one of the members of the set generated by e_g (which should be the metacode of a set generator):

$$\langle \underline{elm}(e_p)\underline{of}\ *(e_g)\rangle \rightarrow \langle \underline{elm}(e_p)\underline{of}\ \langle \underline{step}\ *(e_g)\rangle\rangle$$
$$\langle \underline{elm}(e_p)\underline{of}\ (e_p)e_x\rangle \rightarrow e_p$$

$$\langle \underline{elm}(e_p)\underline{of}\ (e_q)e_x\rangle \rightarrow \langle \underline{elm}(e_p)\underline{of}\ e_x\rangle$$
$$\langle \underline{elm}(e_p)\underline{of}\ \rangle \rightarrow \langle \underline{elm}(e_p)\underline{of}\ \rangle$$

The statement that the proposition *P* logically implies the proposition Q is itself a proposition of our theory, namely:

$$*(\underline{imp}(\uparrow P)\rightarrow\uparrow Q)!$$

3. Knowledge

The cornerstone of our theory is the principle that a mathematical proposition has a meaning only if it can be construed as a (hierarchical) generator of predictions. Until now we have been able to do this easily for various types of propositions. Now we are going to consider two examples of propositions which will show that in order to keep to this principle we have to introduce in our theory a new concept, which to the best of the author's knowledge has never before become part of formal mathematics. This is the concept of *human knowledge*.

The first example deals with *selections*. We saw that the finiteness and the infiniteness of a search find their places in our theory, the former being the quantum of semantics, a prediction, and the latter a generator of predictions. What about the finiteness and the infiniteness of a selection?

To give specific substance to our discussion and relate it to the matters familiar from formal logic, we shall consider those selections which result from the natural constructive interpretation of quantifiers. Let P(x,y) be a total recursive predicate with the set of natural numbers as its domain, i.e. a linguistic machine which initiates a finite process with arbitrary numeric arguments x,y and produces either T or F. Let us see how all possible quantifications of this predicate can be interpreted in terms of processes.

Consider the universal quantification:

(1) $(Ax)P(x)$

To verify this proposition we construct the selection shown in Fig.3.3. We compute P(1), P(2),... etc. and select the end of each computation. If it is F, we stop the process; if it is T, we

go on. Proposition (1) is equivalent to the assertion that this selection is infinite. At the same time, (1) is an infinite generator of propositions P(x) with consecutive numbers x. Each proposition P(x) states that a certain search is finite and produces T, which is easy to represent as a legitimate proposition of our theory.

Generalizing, we can see that the statement of the infiniteness of a selection can always be interpreted as an infinite generator of predictions. At any stage of the underlying process we predict that the current search for a selected stage is finite. These predictions are generated as the process goes on, as in the case of the infiniteness of a process.

To interpret the existential quantification:

(2) $(Ex)P(x)$

we again put computations P(1),P(2), etc. in a row, but this time we simply look for a stage where the result of P(x) is T. Proposition (2) is equivalent to the assertion that the search in Fig.3.4 is finite. It is a prediction.

Quantifying the variables x and y by identical quantifiers does not give anything new, because we can consider the pair (x,y) as one object. Consider mixed quantifications.

To interpret the formula

(3) $(Ax)(Ey)P(x,y)$

we notice that since the predicate

$$P'(x) = (Ey)P(x,y)$$

represents, as we have just established, the finiteness of a search, we can put a universal quantifier on it, as in (1). So proposition (3) will again be the infiniteness of a selection. It is shown in Fig.3.5.

So far so good. Let us now turn to the interpretation of the formula

(4) $(Ex)(Ay)P(x,y)$

By analogy with Fig.3.5, we easily construct a selection which represents the true-false test of this formula; it is shown in Fig.3.6. The selected stages here are those where the computation of the predicate terminates with the value F. We are looking for such a value n of x that the predicate P(n,y) with any y gives the answer T, i.e. the search for F is infinite. Formula (4) is then interpreted as the finiteness of the selection in Fig.3.6. Indeed, if (4) is true, then looking for the y's for which P(x,y) is false with consecutive x's we necessarily come to a stage when there are no such y's; thus the selection is finite.

It is here that a surprise is in store for us. We saw that the assertion of the infiniteness of a selection is a generator of predictions. We expect that the assertion of the finiteness of a selection will also be a meaningful proposition in terms of our theory, i.e. a generator of predictions, of a different kind maybe. But it is not.

The statement that a selection is infinite bears some information about the underlying process. It says that whatever the stage of the process is, there will be a stage later which is recognized as selected. The statement that a selection has no members, or exactly one, or two, or any finite number of members, is also informative and can be interpreted as a generator of predictions. But a statement that a selection is simply finite bears no information which can be used to make predictions. Even with this statement known to be true, whatever the current stage of the underlying process is a selected stage may or may not follow. It cannot be interpreted as a prediction, nor as a process which can ultimately produce predictions. Therefore, we must declare it meaningless.

Yet we cannot deny that intuitively we assign some meaning to the finiteness of a selection and to the equivalent quantified formulas of the conventional formal logic. This meaning, however, includes an element which is left out in formal logic. It is *knowledge*. If we think into what we actually mean when stating that a selection is finite, we shall discover the following mental picture. There is an underlying process, and we start running it. We are looking for a selected stage, and we do not know whether there is one ahead or not. One moment we may discover a selected stage. Then we pass it and go on looking for the next one. Again, we do not know whether there will be one or not,

so we run the process and check stages. But sooner or later we come to a stage when we suddenly -- one way or another -- *get the knowledge* that there will be no more selected stages, i.e. the current search will be infinite. Without this "enlightenment", this phenomenon of getting a knowledge, the assertion of the finiteness of a selection is impossible to understand. It is *implicite* in our intuitive notion of a finite selection. Therefore, in a formal theory it should appear *explicitely*.

In terms of quantifiers, we can interpret a proposition of the form

(Ex)(search S(x) is finite)

by running searches S(∅), S(1), S(2), etc. in parallel. This process will stop if and only if there is at least one value of x for which S(x) is finite. If, however, an existential quantifier is put on a proposition-generator P(x) which produces an infinite set of predictions, then to find the value of x in question we must inevitably employ some way of establishing the truth of P(x), which makes this way a part, though hidden, of the semantics of our assertion. In other words, when we say

(Ex)P(x)

we actually mean

(Ex)(we know that P(x) is true)

whether or not we are prepared to admit it in open.

The second example of a proposition where a reference to human knowledge becomes inevitable is *material implication*. As discussed above, we can impose an if-then connection on any pair of propositions. We shall use the notation

if *P* then Q

for the material implication involving a pair *P* and Q, where *P* will be referred to as the *antecedent*, and Q the *consequent*. How can we formalize this concept?

Consider first the case where the antecedent of an implication is a prediction: 'if the process *A* is finite then proposi-

tion *P'*. There is an obvious way to interpret this proposition as a generator of predictions: we run the search *A*, and when/if it stops, produce proposition *P*. We define the <u>if</u> machine as follows:

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

This machine activates the process e_α, and when and if it ends, produces (e_p) as the result. The auxiliary function <u>second</u> is needed to discard the first element of a list and output the second element. The parentheses in the final result are necessary because a proposition-generator, according to our convention, produces a *list* of propositions (which in this case consists of one member). If the process e_α is infinite, the <u>if</u> machine will go on infinitely, producing no result.

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

$$\uparrow\langle \underline{\text{if}}(\uparrow A)!\underline{\text{then}}\ P\rangle = *(\underline{\text{if}}(\uparrow\uparrow A)!\underline{\text{then}}\ \uparrow P)$$

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

As in the above cases, there is a hidden reference here to a process which establishes the infiniteness of *A*. What we actually mean is "if *we can know that A* is infinite then *P*". In our intuitive understanding of this statement there is no exact definiton of the process through which we get this knowledge. This is no surprise, of course; intuitive understanding is informal. But the vagueness and inexactness of this element of the intuitive picture does not mean that it does not exist or that we should ignore it. We should acknowledge the existence of this element and formalize it, make it exact, one way or another. The first question we have to answer is what is knowledge? How can we formalize this concept?

We define a *knowledge* as a proposition which is believed to be true. This definition reflects our subjective attitude towards knowledge and the way we use it. We use propositions believed to be true in order to make predictions, and we believe in these predictions, i.e. plan our actions under the assumption that

actual processes of the world will conform to the predictions. The question whether what we call our knowledge is actually true is left to an observer (if any) who watches us from outside. The concept of truth is inseparable from an observer, like some fundamental concepts of modern physics. The truth of a proposition is somebody's readiness to plan his actions in accordance with the predictions implied by it. In the absence of this 'somebody', the concept of truth becomes meaningless.

Propositions can be specific or general. Specific propositions concern definite processes at definite historic times. Such a proposition can be checked only a posteriory, when it becomes useless, because it will never be applicable any more. With specific propositions, you are given a chance to verify them only after the question whether you should believe in them becomes meaningless. The question usually posed would be whether one *should have believed* in the proposition before. This question has a pragmatic meaning only if that specific proposition was derived from a certain general proposition; the question then is translated into whether one should believe in this general proposition.

General propositions do not specify a historic time and place of the processes in question, but only the general conditions under which they are applicable. Mathematical propositions are general propositions concerning linguistic processes. They can be partially checked when the question to believe or not to believe is still meaningful. We can verify that a formal prediction is true by initiating the process which it is about and checking that it is, in fact, finite. But a proposition-generator may generate an unending sequence of predictions, and we cannot verify them all. Our readiness to rely on a proposition as true is based, in the last analysis, on a belief, and not on an empirically established truth. We do not discuss at this time the philosophical question of how this belief is arrived at: whether through our experience, or because of the structure of our brain, or as a reflection of a 'higher order' reality. We are exploring the possible ways to formalize the concept of knowledge, and we come to the conclusion that the only thing we can say about our knowledge is that 'we believe because we believe'. Then the only thing we can do in our theory is simply introduce a symbol to denote the sum total of mathematical propositions believed to be true by humanity. We shall use the capital Greek letter Γ (for 'gnosis', knowledge) as such a symbol. We define the machine:

$$\langle \neg e_p \rangle \rightarrow \langle \underline{imp}(\Gamma) \neg e_p \rangle$$

which tries to deduce its argument from the knowledge Γ. If $\langle \neg P \rangle$, where *P* is a proposition, is finite, then *P* is true because it is implied by human knowledge. If it is infinite, then we can say nothing.

By introducing a notation for human knowledge we do not solve all our problems, however. The symbol Γ is a *metasymbol* for the Refal machine; it stands for some expression which we do not (and hardly can) write out explicitely. But here is a problem: the human knowledge does not stay the same; it is developing, growing. Essentially, it is a process, not just an object expression. Then how shall we interpret the concept of truth with respect to this ever changing knowledge?

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

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

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

It is pretty obvious that a consistent theory can be based on the intuitionist viewpoint. Our main effort will be to show how classical logic and set theory based on it work, and to prove their consistency. In this chapter, however, we shall simply use the function γ as a universal way to establish the truth of a proposition, without discussing its nature or how it could have come into existence. This is exactly what we are doing intuitively when we turn over mathematical propositions in our imagination and believe that we understand them.

4. Logical connectives and quantifiers

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

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

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

If $P_1, P_2, \ldots, P_n$ are propositions, then the process

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

will generate all of them and only them. Its metacode

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

is our formalization of the conjunction of the propositions $P_1, P_2, \ldots, P_n$.

Consider two proposition-predictions: $S_1!$ and $S_2!$. (From this notation you can see that the metasymbols S_1 and S_2 stand here for the *metacodes* of the searches, not for the searches themselves). The disjunction of these propositions is the statement that at least one of the two searches is finite. This is the same as to say that the process in which S_1 and S_2 are run in parallel is finite. So we construct the following machine or:

$$\langle \underline{or}(e_1)(e_2)\rangle \;\rightarrow\; s\,|\begin{matrix}\langle \underline{act}\ e_1\rangle \\ \langle \underline{act}\ e_2\rangle\end{matrix}$$

Now the disjunction (S_1 $\underline{or}$ S_2) is formalized as the finiteness of the process

$$\langle \underline{or}(S_1)(S_2)\rangle$$

that is the prediction

$$*(\underline{or}(\uparrow S_1)(\uparrow S_2))!$$

Exercise. Define the generalization of the function $\underline{or}$ which has as its argument an arbitrary list of searches.▽

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

Well, even when putting the meaning in words we cannot avoid a reference to the process of testing the truth of the constituent propositions. We say "at least one of S_1 and S_2 is true". This corresponds to the finiteness of $\langle \gamma\ P_1\rangle$ or $\langle \gamma\ P_2\rangle$ in our formalism. Using the $\underline{or}$ machine we represent the disjunction as

$$\uparrow\langle \underline{or}(\uparrow\langle \gamma P_1\rangle)(\uparrow\langle \gamma P_2\rangle)\rangle!$$

Practice. Convert this expression into strict Refal by eliminating metasymbols ↑.▽

If we use the general formula for disjunction, and one of the operands proves to be a prediction, then a testing process $\langle \gamma\ S!\rangle$, where S is a search, is used, instead of the search S itself. As we shall see later, both ways are completely equivalent.

The material implication was already discussed in Section 3. The proposition $P \rightarrow Q$ of formal logic is represented as

$$\uparrow\langle \underline{if}(P)!\underline{then}\ Q\rangle$$

The property of implication that a false antecedent forms a true proposition with any consequent shocks everyone who studies ma-

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

To formalize quantifiers we need the function of substitution:

$$\langle \underline{sub}(*s_v s_i \rightarrow e_x) : *s_v s_i e_1 \rangle \rightarrow e_x \langle \underline{sub}(*s_v s_i \rightarrow e_x) : e_1 \rangle$$
$$\langle \underline{sub}(e_v) : s_1 e_2 \rangle \rightarrow s_1 \langle \underline{sub}(e_v) : e_2 \rangle$$
$$\langle \underline{sub}(e_v) : (e_1) e_2 \rangle \rightarrow (\langle \underline{sub}(e_v) : e_1 \rangle) \langle \underline{sub}(e_v) : e_2 \rangle$$
$$\langle \underline{sub}(e_v) : \rangle \rightarrow$$

The process

$$\langle \underline{sub}(V^1 \rightarrow E) : P \rangle$$

where V^1 is the metcode of a variable, e.g. *EX (we refer to such expressions as *first-level variables*), and E and P are expressions, results in the expression obtained from P by substituting every entry of V^1 by E.

Using the function <u>sub</u> we define the universally quantifying machine <u>all</u> as follows:

$$\langle \underline{all}(e_v)\epsilon(*(e_g)) : e_p \rangle \rightarrow \langle \underline{all}(e_v)\epsilon(\langle \underline{step}\ *(e_g) \rangle) : e_p \rangle$$
$$\langle \underline{all}(e_v)\epsilon((e_m)e_l) : e_p \rangle \rightarrow \langle \underline{sub}(e_v \rightarrow \langle \bar{\mu} e_m \rangle) : e_p \rangle \langle \underline{all}(e_v)\epsilon(e_1) : e_p \rangle$$
$$\langle \underline{all}(e_v)\epsilon() : e_p \rangle \rightarrow$$

It has the format:

$$\langle \underline{all}(V^1)\epsilon(G):P\rangle$$

where V^1 is a first-level variable, G is a set generator in metacode, and P is a propositional form depending on V^1 (or any expression which may include V^1). The all machine activates the generator G step by step, substitutes the objects produced by G for every V^1 in P, and produces the resulting expression. For instance, if $\langle \nu\emptyset \rangle$ is the generator of all natural numbers, and P is a propositional form representing in our theory some predicate P(x) of formal logic, then

$$\uparrow\langle \underline{all}(E_x)\epsilon(\uparrow\langle \nu\emptyset \rangle):P\rangle$$

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

To express existential quantification we define the searching machine sch:

$$\langle \underline{sch}(e_\nu)\epsilon(*(e_g)):e_p\rangle \rightarrow \langle \underline{sch}(e_\nu)\epsilon(\langle \underline{step}\ *(e_g)\rangle):e_p\rangle$$

$$\langle \underline{sch}(e_\nu)\epsilon((e_m)e_g):e_p\rangle \rightarrow s|\ |\langle \underline{if}(\langle \underline{act}\langle \underline{sub}(e_\nu \rightarrow e_m):e_p\rangle\rangle)!\underline{then}\ e_m\rangle\ |\langle \underline{sch}(e_\nu)\epsilon(e_g):e_p\rangle$$

$$\langle \underline{sch}(e_\nu)\epsilon():e_p\rangle \rightarrow \langle \underline{sch}(e_\nu)\epsilon():e_p\rangle$$

Its format is

$$\langle \underline{sch}(V^1)\epsilon(G):S\rangle$$

where V^1 and G have the same meaning as above, and S is a search in the metacode which may depend on the variable V^1. The sch machine runs the generator G, substitutes the objects produced for V^1 in S and runs all the resulting searches S in parallel. The moment any of these searches comes to an end, the sch machine also stops and outputs the object e_m with which S is finite as its result.

If a predicate P(x) of formal logic can be represented as $P!$ in our theory, the quantified proposition (Ex)P(x) will be represented as

$$\uparrow\langle \underline{sch}(E_x)\epsilon(\uparrow\langle \nu\emptyset \rangle):P\rangle!$$

If P(x) corresponds to a general proposition (generator) *P*, then we must use the cognitive function γ to convert it into a search:

$$\uparrow\langle\underline{sch}(E_x)\varepsilon(\uparrow\langle\nu\emptyset\rangle):\uparrow\langle\gamma S\rangle\rangle!$$

How can we interpret negation? One can think of a *weak* and a *strong* interpretation. The weak negation of a proposition *P* would be simply the exclusion of *P* from the current state of knowledge. We state that we are not sure that all the predictions produced by *P* are true, therefore we do not include it in what we call knowledge. The strong negation of *P* is the assertion that at least one of the predictions generated by *P* contradicts 'the truth', by which we mean a proposition which we include, or will some day include, in our knowledge.

The weak negation is very weak indeed. We cannot infer much, if anything, from it. With a negation defined in this way, we can safely negate even those propositions which are known to be perfectly true. Clearly, it is not this interpretation that is used in mathematics, but the strong version based on the idea of contradiction. So we start with contradiction.

A prediction *A*! contradicts to the truth if *A* is actually infinite, i.e. *A*? is true. But there is no way of testing the truth of *A*? directly. We conclude that, first, the concept of human knowledge must be necessarily present in any formalization of contradiction; second, it is impossible to avoid using the infinity model and express contradiction in terms of predictions only.

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

We can construct a machine which tests that a given proposition is contradictory. Let its format be $\langle\underline{con}\ e_p\rangle$. It can be defined as follows:

#1.1 $\langle\underline{con}\ *(e_g)\rangle \rightarrow \langle\underline{cl}()*(\underline{int}\langle\mu e_g\rangle)\rangle$

#2.1 $\langle\underline{cl}(e_l)(*V(e_p)s(!?)_i)e_g\rangle \rightarrow \langle\underline{c2}\langle\underline{elf}(*V(e_p)\langle\underline{opp}\ s_i\rangle)\varepsilon(e_l)\rangle(e_l)e_g\rangle$

#2.2 $\langle\underline{cl}(e_l)(e_h)e_g\rangle \rightarrow \langle\underline{cl}(e_l)e_g\rangle$

#2.3 $\langle\underline{cl}(e_l)*(e_g)\rangle \rightarrow \langle\underline{cl}(e_l)\langle\underline{step}\ *(e_g)\rangle\rangle$

#2.4 $\langle \underline{cl}(e_l)\rangle \rightarrow \langle \underline{cl}(e_l)\rangle$

#3.1 $\langle \underline{c2}\ T(e_s s(!?)_i)e_z\rangle \rightarrow e_s!?$

#3.2 $\langle \underline{c2}\ F(e_p)(e_l)e_g\rangle \rightarrow \langle \underline{cl}(e_l(e_p))e_g\rangle$

#4.1 $\langle \underline{opp}\ !\rangle \rightarrow ?$

#4.2 $\langle \underline{opp}\ ?\rangle \rightarrow !$

#5.1 $\langle \underline{elf}(e_p)\varepsilon(e_p)e_l\rangle \rightarrow T(e_p)$

#5.2 $\langle \underline{elf}(e_p)\varepsilon(e_q)e_l\rangle \rightarrow \langle \underline{elf}(e_p)\varepsilon e_l\rangle$

#5.3 $\langle \underline{elf}(e_p)\varepsilon\rangle \rightarrow F(e_p)$

Comments.

The idea of the con machine is to run the process $\langle \underline{int}\ e_p\rangle$, where e_p is replaced by the original proposition, and maintain the full list of atomic propositions produced up to date. Recall that the int machine produces directly every proposition which its input proposition can produce hierarchically. Whenever a new atomic proposition is produced it is converted to its opposite and the list of accumulated propositions is scanned in order to determine whether it contains such a proposition.

#1.1 Function con transfers control to the function cl which implements the main recursion of the program. Its first argument is the accumulated list of atomic propositions, which is initially empty. The second argument is the metacode of the int machine. Function con assumes that its argument is a non-atomic proposition (a generator). If it is atomic, no sentence is applicable, which leads to an infinite search (undefined process situation).

#2.1 An atomic proposition generated by $*(e_g)$ is found. Function c2 in the right side calls function elf, 'element of a finite set', to check for contradiction.

#2.2 A non-atomic proposition e_h is discarded.

#2.3 Go on running the generator

#2.4 If the generator turns out to be finite and no contradiction is found, con runs forever.

#3.1 A contradiction is found. The end.

#3.2 The current proposition does not contradict the accumulated list. It is added to the list, and the process continues. ▽

If P is a proposition then we interpret its negation as the statement that the conjunction of P and the human knowledge Γ is contradictory:

$$\uparrow\langle \underline{\text{con}}\ \uparrow\langle \underline{\text{and}}(\Gamma)(P)\rangle\rangle !$$

Introducing the function $\bar{\gamma}$ defined by

$$\langle \bar{\gamma} e_p \rangle \ \rightarrow\ \langle \underline{\text{con}}\ *(\underline{\text{and}}(\langle \mu\Gamma\rangle)(\langle \mu e_p\rangle))\rangle$$

we represent the negation of P as the prediction:

$$\uparrow\langle \bar{\gamma} P\rangle !$$

We have now two 'cognitive' functions: $\langle \gamma P\rangle$ establishes the truth of proposition P, $\langle \bar{\gamma} P\rangle$ establishes its falseness.

5. Prefix notation and free format

Our formal representation of processes and propositions is good for a machine, but a shorter though semi-formal representation based on the usual functional notation will be more convenient for a human being. Further on we shall use such a representation, which in this section is referred to as 'the free format'. In this representation the process initiated by a machine P with an input x is denoted as $P(x)$, the corresponding atomic propositions are $P(x)!$ and $P(x)?$. A process or a proposition will be used as an argument (input) for a machine without explicit reference to the metacode transformation. Thus $\gamma(P(x)?)$ is the process of proving $P(x)?$. The machines we have introduced and corresponding processes will be represented in a more readable form, e.g.

$$\underline{\text{all}}(x\varepsilon S : P(x))$$

Similar notation will be used for the machines to be defined later.

Our 'free format' notation is semi-formal only in the sense that the translation from this notation into strict Refal is defined (in this section) in natural language and not by a program for a machine, so one cannot immediately use a text in the free format as a machine input. Otherwise it is quite formal:

every linguistic object in the free format notation can be in a unique way translated into the strict notation. It should be emphasized that this translation *does not lead* to a significant growth in volume: our strict notation is designed for actual use with computers, and not as a theoretical device only.

The reader who is not interested in knowing how the free format is translated into strict notation can skip the remaining part of this section without detracting from the understanding of our interpretation of mathematics. But the prefix notation has a value -- and, it is believed, a future of its own when it comes to mutual fertilization of mathematics and computer science.

Consider a machine, or function, $\langle Fe_x \rangle$. (As mentioned before, a Refal function and a machine are synonymous concepts). To represent machines as objects of work we have introduced the metacode transformation. In the metacode this machine becomes $*(FE_x)$ (see shorthand notation in Section 2.3 for the explanation of symbols such as E_x). Compare this representation with the usual representation of a function where the function name serves as the prefix of the linguistic object: $F(x)$. We shall make our notation easier to review and closer to habitual standards by introducing *the prefix notation* for Refal objects. We agree that if a machine is defined with a name F and a format which includes variables V_1, V_2, ..., then its format will be represented in the prefix notation as

$$F^0(\uparrow V_1, \uparrow V_2, \ldots)$$

For instance, the format $\langle F(e_x)e_y \rangle$ will become $F^0(E_x,E_y)$ in the prefix notation. The superscript at a functional symbol indicates the metasystem level of the object, i.e. the number of times the metacode transformation has been applied to the original Refal object. The superscript 0 shows that we are dealing with an active Refal expression. When a superscript is absent, 1 is implied, as in algebra. Since propositions are produced by metacoding active Refal processes, the ground level for propositions is 1.

In the prefix notation, we also reserve the right to use separators different from commas -- in particular, those borrowed from the Refal format -- in order to make representation more readable. Thus the format

$$\langle \underline{all}(e_v)\epsilon(e_g):\ e_p \rangle$$

may be represented, by our choice, as

$$\underline{all}^{\circ}(e_v, e_g, e_p)$$

or

$$\underline{all}^{\circ}(e_v \epsilon e_g : e_p)$$

etc., provided that the conversion of this notation into the strict Refal is unambiguous.

We agree, furthermore, that when it does not lead to ambiguitiy we may represent free variables by just their italicized indexes (which must be letters) with metasystem level superscripts, i.e. use x° for e_x or s_x, x for E_x or S_x, etc. The syntax type of the variable is supposed to be known from somewhere. With this convention the metacode of the format above, which represents a propositional form, will be written as

$$\underline{all}(v \epsilon g : p)$$

The substitution of a function format for a free variable in another function format produces a *composition* of functions. The whole diversity of Refal expressions can be produced by substituting object expressions for free variables in function formats and their compositions. However, we have an additional degree of freedom in Refal as compared with the usual functional notation, namely we can apply or not apply the metacode transformation in the process of substitution. Thus we come to two different types of composition, which we refer to as *a call as value* and *a call as process* (a distinction very close to that between a call by value and a call by name in computer programming).

Call as value is the familiar functional composition taught in school. When we write *F(G(x))* we mean that the value of the function G with the given *x* must be computed first, and then the result must be substituted for the argument of function *F*. In our prefix notation, as in the habitual notation, this type of composition is correctly represented by a simple substitution of a function call for an argument. Indeed, according to our agreement, when we write $F^{\circ}(G^{\circ}(e_x))$ it stands for $\langle F \langle G e_x \rangle \rangle$. This machine operates exactly as is required by call-as-value composi-

tion.

Call as process has already been used extensively in this book. The function parg, for example, (see Section 2.6) takes both of its arguments as the metacode of a process, specifically a generator, and instead of just running this process until it ends (in fact, in a typical case it will never end), it interprets the metacode step by step and runs it in parallel with the process represented by the second argument. Here the process itself is important, not only its result in case it stops.

Call as process is the substitution of *the metacode* of the machine being called for an argument of the machine which is calling. If $F^i(A)$ is a function call, then its metacode is $F^{i+1}(\uparrow A)$. When using metasymbols for Refal expressions we shall denote by V^n a variable of level n, and by E^n an object expression of level n.

Consider two machines $F^0(e_x)$ and $G^0(e_x)$, and let the machine G call the machine F as a process. We have here a three-level hierarchy of control, where the machine G (level 3) controls the operation of the machine F (level 2), which in its turn controls the processing of its argument (level 1). There are the following three cases of interaction between level 3 and level 1.

(1)*Value substitution*. The argument of the controlled F machine can be given a definite fixed value, say E. Then the G machine controls a definite process F(E). In the prefix notation we write:

$G^0(F^1(\uparrow E))$ which stands for <G*(F↑E)>

in strict Refal. This is one definite process.

(2)*Variable binding*. The argument of the F machine can be left indefinite, being represented by the free variable e_x of the format. In the metacode e_x becomes a definite object expression, namely E_x. The controlling machine G determines how to treat this object. This is a situation where the G machine deals not with a definite process, but with *a whole machine*. We have:

$G^0(F^1(E_x))$ which stands for <G*(F*EX)>

As in case (1), this is still one definite process.

A good example of this situation is given by the quantifiers all and sch. Consider the expression

$$\underline{\text{all}}^{0}(E_x \epsilon N : P(E_x))$$

where N is the generator of all numbers. The all machine controls two machines here: N, which happens to be just a process (no free variables), and P, which happens to be a parameterized object (no activation brackets but a free variable). The free variable involved is in the metacode; it is an object expression *EX which is used by the all machine as a placeholder showing where to insert the objects produced by running N. Consider the search machine

$$\underline{\text{sch}}^{0}(E_x \epsilon N : S(E_x))$$

where S is a parameterized search: a machine which includes both activation brackets and a free variable. The sch machine substitutes the objects generated by N for *EX in the (metacoded) S machine and runs the resulting searches in parallel. It is one single process, but the S machine with different arguments is forced to take part in it by sch.

(3)*Metasystem reduction*. In this situation a new machine emerges. It is defined in the following way. Give a definite argument E_1 to the F machine and substitute it as a process into the G machine. The result is a definite process. Now give another argument E_2 to the F machine, substitute it into G, observe another process, etc. For every argument of the F machine, the G machine produces a process, thus we have a new machine. What is its formal representation?

With arbitrary object expression E, the process initiated by G is

$$G^{0}(F^{1}(E')) \qquad \text{which stands for} \qquad \langle G*(FE')\rangle$$

where E' is $\uparrow E$, the metacode of E. Therefore the machine in question is:

$$G^{0}(F^{1}(\mu^{0}(e_x)) \qquad \text{which stands for} \qquad \langle G*F(\langle \mu e_x \rangle)\rangle$$

Note that μ has the same metasystem level as G (namely, zero), so

it is called by G *as value*. The function F is in metacode, so it is called *as process*.

For an example of this situation consider the activation machine act in the role of the calling machine G. We have:

$$\underline{act}^0(F^1(\mu(e_x)) \qquad \text{i.e.} \qquad \langle \underline{act}\ {}^*F(\langle \mu e_x \rangle)\rangle$$

This machine takes an arbitrary argument, translates it into the metacode, forms the metacode of machine F with this argument, and activates this metacode, i.e. emulates process F with the given argument. The overall result is that this machine emulates F. If F stops with a certain argument and produces a certain result *R*, then this machine also stops and produces $\uparrow R$.

The three situations we considered can be distinguished by precise syntactical signs. If there are no free variables or their metacodes in the machine called as process, it is situation 1. A first-level variable E_1 or S_1, is a *bound* variable: this is situation 2. A combination $\mu^0(V)$, where *V* is a free variable, points to situation 3.

When we are dealing with propositions all metasystem levels increase by one. The free variable of a machine becomes a first level variable in the proposition based on that machine, be it atomic or non-atomic. So, the statement that a machine $A^0(e_x)$ generates a finite process is $A^1(E_x)!$, or simply $A(x)!$. This is, strictly speaking, not a proposition but a *propositional form*, dependent on a free variable *x* (we shall call first level variables free variables in the context of propositions). A bound variable, which in the machine has the superscript 1, is superscripted by 2 in the derived proposition. Quantifying $A(x)!$ over all numbers *x*, we have:

$$\underline{all}(x^2 \varepsilon N^2 : A^2(x^2)!)$$

Different variables in function formats can be treated differently both in the calling function and in the function being called. To produce all possible results of the composition of the call-as-process type, we first produce all possible combinations of formats and notice that all free variables in the functions being called turn into their metacodes (first level variables). To every of these variables one of the following three substitutions can be applied, which corresponds to the three situations

considered:

	in a machine	in a proposition
Value substitution:	$V^1 \rightarrow E^1$	$V^2 \rightarrow E^2$
No substitution:	bound variable V^1	bound variable V^2
Metasystem reduction:	$V_a^1 \rightarrow \mu^0(V_b^0)$	$V_a^2 \rightarrow \mu(V_b)$

When a composite machine F is called as process by another machine, all the superscripts in the prefix notation of F get increased by one. For instance, let the machine $S^0(x^0,y^0)$ call $P(x,y)$ for y^0 (as process) with the variable x of P bound and y reduced to z^0. (We use letters for variables). The result is:

(1) $$S^0(x^0,P(x,\mu^0(z^0)))$$

Here x^0 and z^0 are free, and x (i.e. x^1) is bound in S. Now let (1) be called as process by yet another machine, $Q^0(x)$. Metacoding and substituting, we have:

(2) $$Q^0(S(x,P^2(x^2,\mu(z))))$$

This is a process, not a machine, for it has no free variables. The variables x^1 and z^1 are bound in Q^0. Still they are free in S^1, as they were in (1); x^2 is bound in S^1, but free in P^2. Generally, in a function F^n of a metasystem level n, variables V^n of the same level are free, while variables V^{n+1} of the next level are bound. Note that variables of different metasystem levels can be denoted by the same letters without any risk of confusion; on this account our formalism compares favorably with the conventional formalism of logic with its necessity of renaming bound variables to avoid conflict. For one who writes a computer program this is a tremendous relief. It is also pleasing aesthetically; renaming variables does not attest to the elegance of a formalism, rather it shows that the details of the machine do not fit perfectly together but require an artificial device to bridge a gap. Suppose we want the Q machine to control machine (1) with variable z^0 in it taking arbitrary values, thus defining a new machine. We can chose any letter to serve as the free

variable for this machine, for instance, x^0 again. We perform the metasystem reduction $z^0 \to \mu^0(x^0)$ in (2), which results in

(3) $$Q^0(S(x,P^2(x^2,\mu(\mu^0(x^0)))))$$

Consider examples of propositions in prefix notation. The set generator *N* of all natural (unary) numbers will be used, as well as the tester = and the adding machine +.

Example 1.
Usual notation:

$$x = y$$

Strict Refal:

$$<=(e_x)(e_y)>$$

is finite with some (unspecified) values of free variables. Prefix notation:

$$=(x,y)!$$

Example 2.
Usual notation:

$$(Ex)(Ay)(y+x = y)$$

First construct the propositional form:

(4) $$=(+(y,x),y)!$$

Note that function = calls function + *as value*: they are at the same metasystem level. Variables *x* and *y* are free here (level 1 in a proposition). Quantifying over *y* we construct the <u>all</u> machine, a generator which produces (4) with all possible *y*'s:

(5) $$\underline{all}^0(y \epsilon N: =(+(y,x),y)!)$$

Its metacode is the corresponding proposition:

(6) $$\underline{all}(y^2 \epsilon N^2: =^2(+^2(y^2,x^2),y^2)!)$$

Here y^2 is bound by the universal quantifier <u>all</u>. But what about

x^2? Syntactically, it is also a bound variable (level 2). What is its meaning?

It has none at the moment. If we do not make a special provision, it will be a semantic error, the case where a machine is given something it does not expect as input. Return to the machine level (5) to see this. Here y is a bound variable, which is replaced, as the <u>all</u> machine works, by specific numbers in the propositional form (4) entering (5). But the free variable *x* remains as it was. As a result, the <u>all</u> machine will produce *predictional forms*, not predictions:

```
                =(+(Ø,x),Ø)!
(6')            =(+(Ø1,x),Ø1)!
                ... etc.
```

If we try to ignore this difference, i.e. do not notice that a free variable is enetring predictions, we face an error in interpretation. Indeed, the first of the prediction says that the process

(7) $\quad <=(<+(Ø)e_x>)(Ø)>$

is finite. But in fact this is not a process, it is a *machine*. We cannot put an expression which contains free variables in the view-field of the Refal machine; first they must be replaced by some values. On the first metasystem level, if we give an argument which includes first level variables to function <u>step</u>, there will be an error, because this function does not expect first level variables; it cannot, generally, make a step uniquely if the contents of the view-field are not fully defined. One can see this comparing (7) with the definition of function +. In order to make a step we must know the value which is to replace e_x.

The general solution of this problem is only to watch that machines get objects of the type they expect as their inputs. With respect to interpretation, we accept the principle that whenever we have an object with free variables, it stands for the whole infinite set of objects obtained from it by substituting every permissible value for every free variable. In the context of propositions this is equivalent to the convention, usual in formal logic, that a free variable is implicitly universally quantified. Then

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

is equivalent to a certain generator, namely

$$\underline{all}^{0}(x\epsilon N:=(+(\emptyset,x),\emptyset))$$

and so are other propositional forms in (6'). The proposition (6) acquires the same meaning as if the bound variable x^2 were quantified universally. (Of course, there is one more hidden convention here: quantification over all possible expressions for x is replaced by quantification over all numbers only).

In order to be on the safe side, we can also avoid interpreting propositional forms as propositions and consider them as a different species: a machine, not a process. Then a generator of propositional forms like (6) is not a legitimate proposition. But we can convert it into a legitimate propositional form, which instead of producing predictional forms (6') produces predictions by replacing x in (6') by a value given to it as an argument. The metasystem reduction of x^2 to level 1:

$$x^2 \to \mu(x)$$

will perform that conversion. Generally, if we are using a propositional form in a call-as-process substitution, every free variable must be either bound by the calling machine or reduced to a free variable in the resulting propositional form. In our case we convert (6) to

(8) $$\underline{all}(y^2\epsilon N^2:\ =^2(+^2(y^2,\mu(x)),y^2)!)$$

To finish the example, we must quantify x existentially. Since (8) is a generator, we first convert it into a prediction by using function ν:

(9) $$\nu(\underline{all}^2(y^3\epsilon N^3:\ =^3(+^3(y^3,\mu^2(\mu(x))),y^3)!))!$$

Here we again had to perform a metasystem reduction in order to keep x as a free variable. Now we simply substitute (9) into the sch machine and take the metacode. This is our result:

(10) $$\underline{sch}(x^2\epsilon N^2:\ \nu^2(\underline{all}^3(y^4\epsilon N^4:\ =^4(+^4(y^4,\mu^3(\mu^2(x^2))),y^4)!))))!$$

Now we make our last step toward a greater freedom in the representation of propositions. We note that if we know about each machine which of its arguments are called as value and which as process, and in each call as process we know which variables become bound and which free, then we can drop all the superscripts, because they can be restored in a unique way. We can also drop all the calls of the functions μ and $\bar{\mu}$ because they can be uniquely restored too. This is provided that we do not use identical variables at different metasystem levels. (Incidentally, this last restriction shows the location of the gap in the conventional notation. One could retort that if superscripts were added to conventional notation to reflect the syntax structure of the proposition then there would be no conflict of variables either. This, however, would again be an artificial device. In contrast, we do not *add* our superscripts. The prefix notation is only a shorter representation of the strict Refal notation, where all objects are constructed by two basic operations: substitution and metacoding. Metacoding has its semantics; it is not just a device to keep track of syntax structure).

Using this agreement, proposition (10) becomes:

(11) $\underline{\text{sch}}(x\epsilon N\colon\ \nu(\underline{\text{all}}(y\epsilon N\colon\ =(+(y,x),y)!)))!$

Further on we shall use this *free format* notation. We shall also allow some further syntactic simplifications, understandable from the context. In all cases when we use a free format it can be uniquely converted into the prefix notation, and then into strict Refal.

Interpretability

1.Real-time and model-time

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

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

Knowledge is the existence in a cybernetic system of a model of some part of reality. Knowledge is both objective and subjective because it results from the interaction of the subject and the object of knowledge. We know that knowledge is never complete.But if the information flow between the subject and the object is only in one direction -- from the object to the subject, we can imagine a complete knowledge (i.e. a complete perfect model) of the object; this idea is not contradictory. If there is a flow of information from the subject to the object

too, the notion of a complete model may become contradictory. Let S_1 be a system-object, and S_2 a system-subject. Suppose S_2 includes a complete model of S_1. Then it can run this model compressing model time with respect to real time, and predict the behaviour of S_1 for all times in advance. This prediction can be sent to S_1, which can change its behaviour so as to falsify the prediction. This contradicts the assumption of a complete model. In particular, the notion of complete self-knowledge (the case when $S_2 = S_1$) is contradictory.

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

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

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

We shall represent real-time processes as follows. Like

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

$$\langle R \rangle \rightarrow E$$

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

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

Access functions to real-time processes in the Refal machine imitate links between our cognitive apparatus and the real world;

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

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

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

```
<gns>  →  Γ
```

the right side of which is unknown to the Refal machine. The functions ν and $\bar{\nu}$ represent the concepts of truth and falsehood. As we mentioned before (see Section 3.3), our formalism allows two interpretations of the concept of truth. One interpretation treats Γ as a definite expression and leads, as we shall show soon, to intuitionist logic. This interpretation is *static* with respect to the real-time process of human knowledge. It does not exclude the possibbility of Γ changing in real time, but during one run of the function ν (or $\bar{\nu}$) the stage of Γ is taken to be fixed, unchanging. The following sentences define the cognitive functions in static interpretation:

```
<ν e_p>  →  <imp(<gns>)→ e_p>
<ν̄ e_p>  →  <con(<μ <gns> >)(<μ e_p>)>
```

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

words, the definition of truth which leads to intuitionist logic is as follows:

Static interpretation of truth

A prediction *A!* is true *relative to* Γ_i (or *in* Γ_i)
if the search *A* with $\Gamma := \Gamma_i$ is finite.
A proposition generator is true *relative to* Γ_i
if with $\Gamma := \Gamma_i$ it produces only propositions
true relative to Γ_i .

The other interpretation corresponds to our intuition of *objective* truth and leads to the usual, classic logic. In terms of our theory it is *dynamic* because it takes the real-time process Γ as a whole and involves all stages Γ_i of Γ in the processes γ and $\bar{\gamma}$. The ~~first~~ former process results from running in parallel the searches

$$\langle \underline{\mathrm{imp}}(\Gamma_i) \rightarrow e_p \rangle$$

with each new stage Γ_i as it appears in real time, and $\bar{\gamma}$ is defined analogously:

$$\langle \gamma\ e_p \rangle \ \rightarrow s \left| \begin{array}{l} \\ \langle \underline{\mathrm{imp}}(\langle \underline{\mathrm{qns}} \rangle \rightarrow e_p \rangle \\ \langle \gamma\ e_p \rangle \end{array} \right.$$

$$\langle \bar{\gamma}\ e_p \rangle \ \rightarrow s \left| \begin{array}{l} \\ \langle \underline{\mathrm{con}}(\langle \mu\ \langle \underline{\mathrm{qns}} \rangle\ \rangle)(\mu\ e_p \rangle) \rangle \\ \langle \bar{\gamma}\ e_p \rangle \end{array} \right.$$

In dynamic interpretation, $\gamma(P)$ and $\bar{\gamma}(P)$ are genuine real-time processes in which the operation of the Refal machine is intertwined with the process of human knowledge. They depend on the time interval Δt that it takes for the Refal machine to make one step, but those features of the cognitive functions which are essential for our theory do not depend on Δt. Let Γ_i be the state of human knowledge at the moment when the Refal machine is making its i-th step. Then the function $\gamma(P)$ will run in parallel the searches:

$$\langle \underline{\mathrm{imp}}(\Gamma_2) \rightarrow P \rangle$$

$\langle \underline{imp}(\Gamma_3) \rightarrow P \rangle$

$\langle \underline{imp}(\Gamma_4) \rightarrow P \rangle$

...

etc., each next process one step behind the preceding. Human knowledge, as we defined it, is a cumulative process, thus the loss of Γ_1 (as well as of any stage except the last in real time) is of no consequence. If ever a stage Γ_i appears in human knowledge such that $\langle \underline{imp}(\Gamma_i) \rightarrow P \rangle$ is finite, and only in this case, the search $\gamma(P)$ will be finite. The search $\bar{\gamma}(P)$ will be finite if and only if P contradicts some stage Γ_i.

The corresponding definition of truth is:

Dynamic interpretation *of truth*

A prediction $A!$ is true if there is such a (true) knowledge Γ_i that, with $\Gamma := \Gamma_i$ the search A is finite.
A proposition generator is true if with any (true) knowledge Γ_i it produces only true propositions.

The possibility of representing processes by objects through the use of the metacode and activating them through the use of the functions <u>step</u> and <u>act</u>, when necessary, has been an essential part of our formalism. When we allow real-time processes like $\langle \underline{tempNYC} \rangle$ and $\langle \underline{gns} \rangle$, functions <u>step</u> and <u>act</u> become partially undefined. We extend the definition of these functions by expressing their result through the access functions for real-time processes. Let $\langle R \rangle$ be a real-time process (we shall often identify a real-time process with the corresponding access function in order not to encumber our terminology). The result of the concretization of $\langle \underline{step} *(R) \rangle$ should be the metacode of the result of one step of the concretization of $\langle R \rangle$. Since we postulate that access functions are concretized in one step, we define the <u>step</u> machine when applied to a real-time process $\langle R \rangle$ by the sentence

$$\langle \underline{step} *(R) \rangle \rightarrow \langle \mu \langle R \rangle \rangle$$

The function <u>act</u> gets defined thereby, but we can also add one more sentence for the reader's convenience:

$$\langle \underline{act}\ *(R)\rangle \rightarrow \langle \mu\ \langle R\rangle\ \rangle$$

These sentences are presumed to be found in the memory field for every real-time process.

Metacoded expressions representing processes fall now in two categories:

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

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

The process $\langle \gamma\ P\rangle$, which we denote simply by $\gamma(P)$ in the free format notation, is an example of a metamechanical process. So is $\bar{\gamma}(P)$. Let us trace how real-time access functions are called when we deal with metamechanical processes represented in metacode. Suppose we activate $\gamma(P)$, that is place

$$\langle \underline{act}\ *(\gamma P)\rangle$$

in the view field of the Refal machine. The first branch will become:

$$\langle \underline{act}\ *(\underline{imp}\ (*(\underline{gns}))\rightarrow P)\rangle$$

then

$$\langle \underline{act}\ *(\underline{imp}(\langle \underline{act}\ *(\underline{qns})\rangle) \rightarrow P)\rangle$$

and then

$$\langle \underline{act}\ *(\underline{imp}(\langle \mu\langle \underline{qns}\rangle\rangle) \rightarrow P)\rangle$$

which requires an access to human knowledge for further concretization.

2. 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 we shall define later.

A *formal system* is defined if:

(1) a Refal representation for a number of parametrized processes are 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 sytem 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 γ_i and $\bar{\gamma}_i$. Thus a proposition P is provably true in F_i if and only if $\gamma_i(P)$ is finite. It is provably false if and only if $\bar{\gamma}_i(P)$ is finite. The set generator $\underline{int}(\Gamma_i)$ produces all propositions provable in F_i.

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 produced by $\underline{int}(\Gamma_i)$, it corresponds to the derivability of P from Γ_i in the 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. Intuitively, every prediction $A!$ is either true or false: true if A is finite and false otherwise. Every proposition is also either true or false: it is true if and only if it hierarchically produces only true predictions. This intuition, which includes the idea of potential infinity, is the basis for proofs in our metasystem. Once a formal system is created, its further use is, as in the case of usual syntactic systems, purely mechanical. Intuitive proofs we construct in the metasystems serve to justify the formal systems we create.

We argue that this co-existence of a metasystem which has formal objects and intuitive proofs, with formal systems in which proofs are mechanical is not only natural but necessary. Formal systems are created in order to express and produce knowledge. But how do we know that all the propositions that can be mechanically produced in a certain formal system F_i are true? As we already discussed, we have no way to know it for sure; we can only trust to our intuition that a given system can be relied upon. Moreover, we shall be reluctant to use F_i if our intuition does not suggest that F_i is *correct*, i.e. produces only true propositions. And we shall never use F_i if we feel intuitively that it is not correct. Of course, our intuition is not infallible, and it may happen that a formal system we have constructed yields a false prediction. Then we make a change to eliminate the error; the experience of an error and its correction leaves an imprint on our intuition. Still intuition remains the supreme judge in the construction of formal systems.

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. Sometimes we say 'a theory' meaning, in fact, the formal system which is the latest stage of a theory. 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:

Ø, Ø1, Ø11, Ø111, ... 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:

$$\langle +(e_x)(\emptyset)\rangle \;\rightarrow\; e_x$$

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

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:

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

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; its well-known properties, stated as axioms, is all we know about it.

The only way to satisfy our intuition that a given non-trivial formal system is trustworthy is to construct it starting from scratch and proceeding by steps in such a way that it is intuitively convincing that if our formal system was correct before such a step, it will be also correct after the step. The most obvious way to make a step is to add a new proposition to the current knowledge. This proposition may be the formalization of a prediction proven intuitively. Or it may be a generator of predictions; then an intuitive proof should be given that it generates only true predictions. In both cases we must be sure that after adding a true proposition to a correct system we receive a correct system: an assumption which, as we shall see soon, is not automatically true.

The act of adding a proposition to the knowledge of a formal system is a *statement* . Using a notation fashioned after the programming language ALGOL, the statement P is:

(1) $\Gamma := \underline{\text{and}}(\Gamma, P)$

A *correct* statement is such that the resulting formal system is correct if the original system is correct.

We must now find a firm ground on which to construct intuitively safe proofs that a given proposition is true or false. Since the definitions of truth are different in intuitionistic and classical logics, we examine the grounds for these two cases separately. Nevertheless, we shall come to the same restrictions on the use of cognitive functions in both logics. In this section we take up the static, intuitionist, definition of truth.

With the static definition, the searches γ and $\bar{\gamma}$ become mechanical after the second step. For a mechanical search it is intuitively safe to believe that it is either finite or infinite. If the current human knowledge Γ_i is consistent, and we always assume that it is, then $\gamma(P)$ and $\bar{\gamma}(P)$ cannot be finite simultaneously. Therefore, three situations are possible:

$\gamma(P)$! and $\bar{\gamma}(P)$?	P is true
$\gamma(P)$? and $\bar{\gamma}(P)$!	P is false
$\gamma(P)$? and $\bar{\gamma}(P)$?	P is neither true nor false

This can serve as the basis for assigning truth-values to propositions. There is a snag here, however. When Γ_i changes, the functions γ and $\bar{\gamma}$ change too, and a proposition which was marked true before can now become false. We certainly do not want that. Specifically, when we construct a formal system by steps of the form (1), our knowledge may grow, and this will change the current human knowledge Γ_i. Therefore, we must examine in greater detail the relationship and interaction between the human knowledge referred to by propositions and the knowledge of the formal system. In the most important case the two coincide. This will take place if we consider the real-time construction of a formal system which fixates our real knowledge in some field of mathematics. Then we simply have no pertinent knowledge other than the one put in Γ_i at every moment in time. So, the same sequence Γ_1, Γ_2, Γ_3,... etc. represents both the steps of the construction of the formal system, and the stages of pertinent human knowledge.

Let the knowledge of the current system be Γ_i. There are two ways of understanding (1). We can understand it as a sort of 'evaluation' of knowledge, meaning by the 'value' of a knowledge its meaning, i.e. the hierarchical set of predictions it produces. Then to produce Γ_{i+1} we must first 'evaluate' P, using Γ_i, i.e. substitute Γ_i for human knowledge in P, and then add the resulting knowledge to Γ_i. Suppose P is

$$\underline{\text{if}}\ \gamma(P)\,!\,\underline{\text{then}}\ Q$$

and remember that $\gamma(P)$ is by definition

$$\underline{\text{imp}}(\Gamma \rightarrow P)$$

Then

$$(2) \qquad \Gamma_{i+1} = \underline{\text{and}}(\Gamma_i, \underline{\text{if}}\ \underline{\text{imp}}(\Gamma_i \rightarrow P)\,!\,\underline{\text{then}}\ Q)$$

Using this approach systematically we have to keep track of all the previous formal systems in our current formal system. Moreover, it contradicts our intuitive understanding of conditional propositions. When we say "if P is true" we mean "true according to the sum total of our knowledge", therefore the use of Γ_j with j less than the index of the current formal system in the *if* clause is not justified. It can be shown that if we accept this approach to statements, some fundamental formulas of logic common to both classical and intuitionist logic (specifically, the contraposition law) will not hold.

Therefore we adopt the other understanding of (1), according to which the operation *and* in (1) includes no 'evaluation' of knowledge, but is performed over the formal representation of propositions, not over their meanings. In each formal system resulting from consecutive statemants we use the same symbols γ and $\bar{\gamma}$, which refer to the knowledge Γ of the formal system itself. According to the definition of functions *imp* and *con*, this means that γ and $\bar{\gamma}$ may call themselves recursively. For the example above, this approach gives

$$(3) \qquad \Gamma_{i+1} = \underline{\text{and}}(\Gamma_i, \underline{\text{if}}\ \underline{\text{imp}}(\Gamma \rightarrow P)\,!\,\underline{\text{then}}Q)$$

where Γ_i and Γ_{i+1} are now *formulas* which may include (in fact, almost certainly do) reference to Γ through the functions γ and $\bar{\gamma}$. The moment we execute (1), the cognitive functions in both P and Γ start referring to the new knowledge Γ_{i+1}.

This recursive nature of formal systems, however, leads to certain paradoxes. When we make a statement we change the formal system. Since the proposition we are adding may refer to cognitive functions, and they depend on the knowledge of the current system Γ_i, which changes in the process of addition, the meaning

of the proposition when it becomes a part of system may be different from what it was before. A proposition *P* could be true, but the statement *P* incorrect.

We can show it in this simple example. Let *A* be an infinite process and suppose that the current formal system is such that it cannot establish the infiniteness of *A* , i.e. the process of proving *A*? is infinite: $\gamma(A?)?$. Then the proposiiton

(4) $\quad P = \underline{and}(A?, \gamma(A?)?)$

is true. However, the moment we add *P* to the formal system, the knowledge Γ is changed and includes now *P*; the proposition *A*? becomes provable, and $\gamma(A?)?$ false . The new Γ is, therefore, false.

One might think that this paradox occurs because of the inner contradiction in the proposition (4). It is true that (4) will create an incorrect system when added to any, even an empty, formal system. But consider a formal system Γ_1 which includes $\gamma(A?)?$ and nothing more. It is correct. Proposition *A*? is safely true. Still when we add the two we again have an incorrect system.

To be able to construct correct formal systems by adding true propositions, we must put forward certain additional requirements to the propositions used in statements. The current human knowledge in static interpretation can be called the *context* in which a proposition acquires an exact meaning and a truth value. The paradoxes we have just seen arose because the proposition $\gamma(A?)?$ which had been true in the original context, became false when *A*? (true in itself) was added to the context. Therefore, what we need in propositions can be called *the context safety* .A proposition *P* is *context safe* if once found true (false) in a true context Γ_i, it will also be found true (false) in the context of the conjunction $\Gamma_i \& \Gamma_j$ with any true Γ_j.

3. Strong Interpretability

Propositions of our theory should deal with context-safe processes only. This is a necessary condition. But we want something else. According to our naive intuitive interpretation of mathematical propositions they do not refer to any human know-

ledge at all, but express "objective" properties of processes. We take issue with this view and offer a more formal interpretation which makes an explicit reference to human knowledge and its *subject* in the philosophical sense, i.e. the possessor and developer of that knowledge -- the human race. But we do not want to throw away our intuition of objectivity, rather we want to reinterpret it in our terms. We shall limit ourselves to such propositions only that allow an intuitively objective interpretation, or rather, evaluation, which assigns to it one of two truth-values: T (true) and F (false). We call these propositions *interpretable*. The definition of interpretability which we are going to give does not depend on the definition of the functions γ and $\bar{\gamma}$; it is equally good for intuitionist and classical logic. But it is derived from the classical, not intuitionistic, logic. The interpretation which stands behind the concept of interpretability is dynamic, classical. The definition of interpretability will make an interpretable proposition automatically context-safe.

Interpretability is defined 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 U is such a generator, and none of the propositions it produces refers to cognitive processes, then U is interpretable.

The role of the cognitive processes γ and $\bar{\gamma}$ in our theory is to establish that a given proposition is intuitively true or false. Implementations of γ and $\bar{\gamma}$ which are different but always have the same effect (i.e. stop or do not stop) for any given argument should be considered equivalent. Specific stages through which a cognitive process passes should be of no consequence. This is a necessary condition for context-safety. Indeed, as new knowledge is added to Γ, every stage of the processes γ and $\bar{\gamma}$ changes; still we expect that $\gamma(P)$ will stop only if P is true, while $\bar{\gamma}(P)$ will stop only if P is false. We create cognitive processes in order to predict the behavior of some underlying "objective" processes, and for our theory to reflect "objective" reality we must be able to interpret every proposition in terms of objective processes only, without reference to any particular stage of the process of cognition.

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 be-

comes ingrained in the meaning of the equations. Yet the most important physical quantities are those which are *invariant* with regard to transformations of the reference system. 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 mathematcal knowledge. The analysis we made in Chapter 3 showed 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.

Our choice of logical machines is arbitrary (although very understandable), and so is the choice of the generators Γ_i representing our knowledge. Moreover, we are not limited only to those cognitive functions which are expressed through the machines imp and con; nor are we limited to just two cognitive functions. A general theory of knowledge, the metatheory of scientific theories, can consider an arbitrary number of cognitive functions and look for invariants independent of them. In our theory we have two cognitive functions γ and $\bar{\gamma}$, which 'measure' the truth-values of propositions. Interpretable propositions are analogous to invariants of physics: their truth-values do not depend on the choice of cognitive devices.

Consider the proposition $\gamma(P)!$, where P is a proposition from the inductive base, i.e. not referring to cognitive processes. Its meaning is: using certain means we have been able to prove that P is true. If we do not say anything about the means used, this meaning is the same as if we simply stated the proposition P. The proposition $\gamma(P)!$ has the same objective interpretation as P.

Consider the proposition $\gamma(P)?$. It means: using certain means we are not able to prove P. Unlike the case of $\gamma(P)!$, this proposition loses its content if we do not specify the means

used. It does not tell us anything about the truth or falsehood of P, it has no objective interpretation. Moreover, the proposition $\gamma(P)?$ directly violates the requirement of context-safety as formulated above. It stands for the generator $\underline{\inf}(\gamma(P))$, which is a generator of predictions explicitely involving every stage of the process $\gamma(P)$. Therefore we ban propositions of the form $\gamma(P)?$ from our theory, while allowing $\gamma(P)!$. By the same reasoning we allow $\bar{\gamma}(P)!$ but ban $\bar{\gamma}(P)?$.

Generalizing this argument, 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 below as *weak* interpretability.

The 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 $\gamma(P)$ or $\bar{\gamma}(P)$, then if
(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). ∎

We proceed now to analyze our intuitive belief that a meaningful proposition is either true or false -- *objectively*, i.e. independently of whether we exist or not. We do not accept this notion literally for the reasons discussed in Chapter 1 and above in this section. We want to interpret it in terms of our theory.

The fundamental role in this interpretation is played by the principle of the excluded middle in the form: Every mechanical search A is either finite or infinite:

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

Our intuition accepts this principle without hesitation; it seems impossible to deny it. But its formulation (EM_1) in our theory may seem questionable, which, in turn, may put in question our main thesis. One might ask:

(a) How do we know that the human race will accumulate knowledge infinitely? Is it not possible, or even certain, that at some time humanity will cease to exist? Then Γ will stop changing in real time, and the proposition $A?$ for some infinite A may never be proved.

(b) Even if we assume that the process Γ is infinite, is it not possible that such a process A exists that, though it is actually infinite, we shall never be able to prove it?

The answer to the first question is: of course, we do not know whether humanity will accumulate knowledge infinitely. But it has nothing to do with our theory. We are engaged in metamathematics, not futurology. Humanity is *the subject*, not *the object* of our study, as it is in futurology. Its role, or mode, is different. The modality of model-time processes is necessity. We say 'this process is finite' meaning that it will necessarily stop. The future of such processes is predetermined, definite. The modality of real-time processes is always possibility, never necessity. Even when we say about a real-time process, R, which is the object of our study that it is finite, the concept 'finite' is actually applied to the model time. To state that R is finite is to create some model M of it, a mechanical linguistic process which is finite, and then use this model in decision making as a substitute for R.

The real-time process of human knowledge in our theory is not an object of study: our object of study is the linguistic processes we create in mathematics. The human knowledge process is completely in the mode of possibility. We do not create its models, we *participate* in it, we *make it up*. The correct reading of $\nu(P)!$ is not that this process *is* finite, or *will* come to an end, but that it *can* come to an end. According to our defintion of dynamic interpretation of truth, the statement $\nu(P)!$ means that such a stage Γ_i of human knowledge is possible which implies P. If I can indicate such a Γ_i (which is accepted as true by my intuition, this is always a necessary condition), then I consider the matter decided. But I can have no convincing Γ_i to prove P and still discuss the question whether such a Γ_i exists or not. Then $\nu(P)$ becomes a part of my theory without my knowing whether it is finite (Γ_i exists) or infinite (it does not exist). The meaning of 'exists or not' with respect to the cognitive functions is, again, subjective, not objective. It is essentially a *mindset*. Since the process of cognition is not given objectively, its future depends on what decisions we are taking now. To state that the Γ_i in question exists, without actually presenting and justifying it, means to set the goal of finding it. It may happen that such a statement will become a self-fulfilling prophecy. If we were trying to be objective, such occurencies should have bothered us. But we are not after an objective study of Γ; it is impossible. A statement which refers to cognitive functions is, before it is proved or disproved, a plan of cognitive action, a proposal to act, a *proposition*. It is we ourselves who define our cognitive process. We are not completely free to define it, because the propositions we declare as knowledge must not contradict our experience and must satisfy our intuition as true. But within these limits we *are* free. If we decide to set an additional limit to our knowledge, this limit becomes a reality, a self-fulfilling prophecy. To expand our knowledge in the maximal possible way, we should accept as true any proposition which does not lead to contradiction. Who asks more, gets more. We must accept as poossible everything that is not proved impossible.

The answer to the question (b) is that the way it is asked contains a contradiction. As we discussed more than once, the statement 'the process A is actually infinite' is true if and only if $\nu(A?)$ is finite. There is no way to verify this statement other than to refer to the process of human knowledge. This gives us a hint that (EM_1) can never be proved false. Indeed, to prove that the process or in (EM_1) is infinite we must prove that

A is infinite. The moment we do it, we prove that $\gamma(A?)$ is finite. Then the or process is finite. So, (EM_1) cannot be false. This proposition expresses a mindset, namely, our determination to explore the process *A* until we either discover its end or prove that it is endless. There is no third possibility, because we do not want it: this is the origin of the principle -- or the law -- of the excluded third (middle). The intuiitionists reject the law of the excluded middle. They are free to do it. But one must uderstand the nature of this decision. It is putting a roadblock on the path of knowledge without any compelling reason for it.

What we have just proved can be formulated as

Lemma. The assumption of (EM_1) cannot lead to contradiction.

Our intuitive conviction that every search is either finite or infinite is based on the acceptance of (EM_1). Although on the surface of it no process is evoked when we say '*P* is actually either finite or infinite', the meaning of this expression includes a cognitive process, as every statement does: knowledge is interaction of the subject and the object.

We shall prove now that for every interpretable proposition we can construct a process of objective evaluation which will lead to marking the proposition as either true or false.

(a)Base. Consider an atomic proposition *A*!, or *A*?, where *A* is a mechanical search. We believe that such a proposition "actually is" either true or false. But what does it mean? That there is a process which results in marking the proposition as true or false. This process can be constructed using (EM_1). It consists in running the searches *A* and $\gamma(A?)$ in parallel and marking the proposition T or F depending on which branch ends. We call this process *the objective evaluation* of a proposition:

$$\langle \underline{obj}\ e_\alpha! \rangle \rightarrow s| \begin{array}{l} |\langle \underline{true} \langle \underline{act}\ e_\alpha \rangle \rangle \\ |\langle \underline{false} \langle \gamma\ e_\alpha? \rangle \rangle \end{array}$$

$$\langle \underline{obj}\ e_\alpha? \rangle \rightarrow s| \begin{array}{l} |\langle \underline{false} \langle \underline{act}\ e_\alpha \rangle \rangle \\ |\langle \underline{true} \langle \gamma\ e_\alpha? \rangle \rangle \end{array}$$

$\langle \underline{\text{true}}\ e_x \rangle \rightarrow T$
$\langle \underline{\text{false}}\ e_x \rangle \rightarrow F$

All we said above concerning the disjunction in (EM_1) is applicable also to the process obj. In particular, the Lemma guarantees that the assumption of the finiteness of obj will not lead to a contradiction.

(b)Induction on generation. Consider a proposition-generator which calls no cognitive functions; let it be *P*. We can assume that *as a process* it is infinite. If it 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 and mark the proposition-genrator as F. If it is T, we go on running the generator. This defines a certain process; let us denote it as *A*. Although *A* is not mechanical, the reasoning which led us to accept (EM_1) is still applicable to it. Again, the assumption of (EM_1) cannot lead to a contradiction. Therefore we construct the finite process of objective evaluation which runs the process *A* and the proof of its infiniteness in parallel. This process assigns to *P* a definite truth-value.

(c)Induction on cognitive function calls. Consider an interpretable search, say *A*. Whenever $\gamma(P)$ or $\bar{\gamma}(P)$ is called, the proposition *P* is interpretable and, by the induction hypothesis, has an objective evaluation. Modify *A* as follows. When $\gamma(P)$ is met, initiate the process of objective evaluation of *P*. It is always finite. If the result is T, replace $\gamma(P)$ by any object expression (according to point I.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'*. It is either finite or infinite. In the first case we intuitively take the proposition *A!* as true, in the second case as false. Reasoning as in case (b), we can apply (EM_1) to *A'*. The process obj(*A'!*) must be finite. It is the objective evaluation of *A!*. It results in the assignment of a definite truth-value to it.

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

A process, say *A*, is *semantically dependent* on another process, say *B*, if one of the future stages of *A* includes $\langle\gamma\uparrow B!\rangle$, or $\langle\bar{\gamma}\uparrow B?\rangle$, or $\langle\bar{\gamma}\uparrow B!\rangle$, or $\langle\bar{\gamma}\uparrow B?\rangle$ as a subexpression. For the process *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*. The relation of semantic dependence is obviously transitive. 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, e.g., the process <A> defined by the sentence:

(1) $\langle A\rangle \rightarrow \langle\gamma^*(A)?\rangle$

is not interpretable. Compare it with the usual infinite recursion:

(2) $\langle A\rangle \rightarrow \langle A\rangle$

The process <A> defined by (2) is perfectly interpretable and infinite. The question whether <A> defined by (1) is finite or infinite has no meaning. We understand what finiteness/infiniteness is when we speak of deterministic, mechanical processes. With real-time processes, as we discussed above, the modality of necessity gives place to the modality of possibility. The concept of finiteness/infiniteness loses its meaning unless it is somehow defined. For interpretable processes we can construct such a definition through the objective evaluation. For uninterpretable processes we have no definition.

If we assume that the process defined by (1) can be judged in terms of finiteness/infiniteness, we immediately come to a contradiction. Indeed, if <A> is finite then the right side of (1) is also finite, which means that <A> is infinite. If <A> is infinite then $\langle\gamma^*(A)?\rangle$ should be finite, which makes <A> finite.

Uninterpretable propositions stand behind all paradoxes of mathematics. Note that if we change (1) in this way:

(3) $\langle A \rangle \rightarrow \langle \gamma^*(A)! \rangle$

we will not be able to come to a contradiction -- not immediately at least -- but $\langle A \rangle$ defined by (3) is still uninterpretable. ∇

Objectve interpretation has double importance. First, it gives an instrument to decide which proposition is true and which is not. According to the general definition of truth in the dynamic interpretation of cognitive function, in order to add some proposition, say *P*, to human knowledge, we must prove to the judge of intuition that *P* will hierarchically produce only true predictions in the context of any knowledge. How can we prove it in a convincing way? The definition of objective interpretation reduces this problem to a series of problems of one special type: a proof that a given search is infinite. If we know how to solve this problem -- and to the extent we are able to solve such problems -- we can solve any mathematical problem.

Second, objective interpretation divides all interpretable propositions into two categories: true and false. This is a tremendous help in analyzing complex propositions. The concept of falsehood becomes defined on its own, independently of the concept of contradiction. Yet, as we shall establish in the next section, contradiction and falsehood go together, so we still need only one function $\bar{\gamma}$ to recognize them.

4. Weak interpretability.Correctness theorem

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.

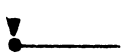
When examining an uninterpretable branch, the function obj will work infinitely in a futile attempt to get to the bottom of semantic recursion; meanwhile, another branch may lead to a definite result.

For a better insight into the structure of propositions, we shall use their *semantic maps*. The function obj will be defined as the process of labeling the semantic map of a proposition.

The following is the definition of the elements of semantic maps.

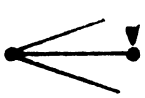
A proposition-generator which starts producing something. A semantic map is a directed graph with nodes (dots) representing propositions and arcs (lines) representing processes. Unless the direction of an arc is indicated explicitly, it is from left to right and from top down. If a line peters out, it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

A proposition-generator which branches into three parallel processes.

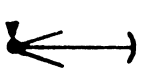


A proposition-prediction that a search is finite; and the beginning of that search.

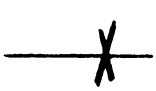
A search branches into two parallel searches.



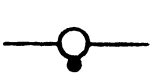
A branch of a generator produces a prediction.



A branch of a search ends.

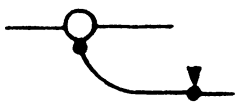


A process becomes infinite. The part to the right of the cross never materializes.



A process calls $\gamma(P)$ and goes on. P is represented

by a dot, i.e. as a proposition-generator.

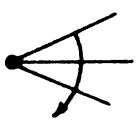


A process calls $\gamma(A!)$ and goes on. We do not distinguish between a proposition-generator which generates one prediction, and that prediction.

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

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

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



A process has more branches than shown in the map.

An infinite loop in the process.

Examples of semantic maps

Fig.1. Proposition ($A!$ and $B!$) where A is finite and B infinite.
Fig.2. Proposition if $\gamma(B?)!$ then or($\bar{\gamma}(A!)$,$\bar{\gamma}(A?)$)!
with A finite and B infinite.
Fig.3. Proposition if $\bar{\gamma}(P)!$ then $\bar{\gamma}(P$ and $Q)!$,
where P and Q are propositions whose truth-values are as yet unknown.
Fig.4. Proposition isr! where isr is defined by:

$$\langle \underline{\text{isr}} \rangle \rightarrow \langle \gamma\ *(\underline{\text{isr}})? \rangle$$

There are a number of simple transformations that can be done over a semantic map 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.

A semantic map can be *labeled*. This is a process which results in the marking of every $\gamma/\bar{\gamma}$-call as either finite or infinite and the assignment to every proposition in the map of one of the following truth-values: **T** for true, **F** for false, and **U** for uninterpretable. Labeling the map of a proposition *P* is the same as giving it objective interpretation. There is no algorithm that could label the semantic map for every proposition; the process of labeling is metamechanical. We have no general method to *determine* the labeling; we only *define* it.

The rules of labeling are as follows.

Labeling rules

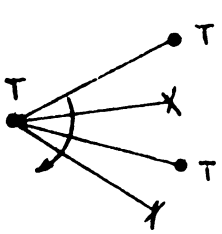
LR1. A call $\gamma(P)$ with *P* labeled **T** is marked finite.

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

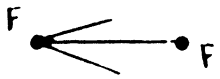
LR3. A call $\bar{\gamma}(P)$ with *P* labeled **T** is marked infinite.

LR4. A call $\bar{\gamma}(P)$ with *P* labeled **F** is marked finite.

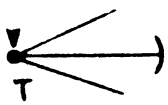
LR5. A call $\gamma(P)$ or $\bar{\gamma}(P)$ with *P* labeled **U** is left unmarked.



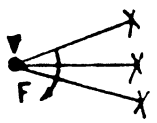
LR6. If every branch starting from a proposition-generator either leads to a proposition labeled **T**, or is infinite, this proposition-generator is labeled **T**.



LR7. If at least one branch starting from a proposition-generator leads to a proposition labeled **F**, then this proposition-generator is labeled **F**.

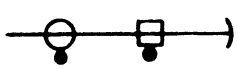


LR8. If at least one search starting from a prediction node is finite, then this prediction is labeled **T**.

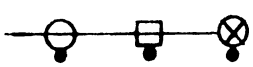


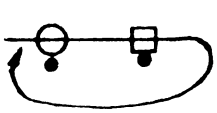
LR9. If every branch starting from a prediction node is infinite, then this prediction is labeled **F**.

LR10. A proposition which cannot be labeled by the rules above is labeled **U**.



LR11. A branch is finite if it ends, and all $\gamma/\bar{\gamma}$-calls on it are marked finite.





LR12. A branch is infinite if one of these cases takes place:

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

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

Examples of labeling. Fig.5 shows the map of Fig.2 after labeling. Fig.6a and Fig.6b show the labeling of the semantic map of Fig.3 with *P* true and *P* false. In both cases the root proposition is labeled true, no matter what the labeling of Q is. The prediction in Fig.4 must be labeled **U**. Infinite semantic recursion is not necessarily the result of a situation when a stage of the process exactly repeats itself. We can define function isr this way:

$$\langle \underline{isr}\ e_x \rangle \rightarrow \langle \gamma\ {}^*(\underline{isr}\ \langle \mu e_x \rangle)? \rangle$$

Then for <u>isr</u>()! we shall have the semantic map shown in Fig.7.

Rules LR7 and LR8 do not require that all branches are labeled definitely (i.e. **T** or **F**): 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. We shall see later that weakly interpretable processes play an important role in set theory.

Consider the paths in the semantic map of a proposition *P*, which start at node *P*. When such a path passes over from a call $\gamma(Q)$ or $\bar{\gamma}(Q)$ to Q we have a *metasystem transition*. The number of metasystem transitions on a path is its *semantic length*. A path is *semantically finite* if its semantic length is finite, and *semantically infinite* otherwise. A path is *unavoidably semantically infinite* if it is semantically infinite, and (1)whenever it passes through a proposition-generator there is no other branch starting from the same generator such that none of the paths taking this branch is unavoidably semantically infinite, and the branch produces a proposition labeled **F**; (2)whenever it passes through a prediction ,there is no other branch such that none of the paths taking this branch is unavoidably semantically infinite, and this branch comes to a halt (is finite). A proposition is strongly interpretable if its map allows no semantically infinite paths. A proposition is weakly interpretable if no path in its map is unavoidably semantically infinite. It is possible that a given map contains finite paths of arbitrarily large semantic length.

Note an essential difference between true and false propositions when we allow weak (partial) interpretability: a false proposition-generator can produce uninterpretable propositions, while a true one cannot.

When we create a formal system we take a proposition Γ_i as its knowledge and define the access function <u>gns</u> as

$\langle \underline{\text{gns}} \rangle \rightarrow \Gamma_i$

Now the function $\gamma(P)$ called by proposition-generators, which was undefined before, becomes a completely defined recursive function which we denote as $\gamma_i(P)$; and $\bar{\gamma}(P)$ becomes $\bar{\gamma}_i(P)$. Note that the replacement of $\gamma/\bar{\gamma}$ by $\gamma_i/\bar{\gamma}_i$ takes place only for the purpose of generation. The ultimate product of proposition-generators, the predictions, can still include $\gamma/\bar{\gamma}$-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 $\gamma(P)$ and its 'approximation' $\gamma_i(P)$. What we want, of course, is that the formal system be correct, i.e. $\gamma_i(P)$ be finite only when $\gamma(P)$ is finite (P objectively true). This relationship is established in the following theorem, which is crucial for the whole theory we are developing.

Correctness theorem. If Γ_i is true then $\gamma_i(P)$ 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 $\gamma_i(P)$ be finite and suppose that P is false. The process $\gamma_i(P)$ 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 map of Γ_i. They are different only in that every call $\gamma(Q)$ in B is replaced by $\gamma_i(P)$ in $B_i(P)$, and every $\bar{\gamma}(Q)$ is replaced by $\bar{\gamma}_i(P)$. The branch B_i has no more than a finite number of $\gamma_i/\bar{\gamma}_i$ calls. Let them be:

$$(*) \qquad \gamma_i(Q_1),\ldots,\gamma_i(Q_n),\bar{\gamma}_i(Q'_1),\ldots,\bar{\gamma}_i(Q'_m)$$

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 (Γ_i 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 $\gamma_i/\bar{\gamma}_i$ calls they involve (if any). Since the process of producing P from Γ_i is finite, we shall ultimately come to a finite *derivation tree* for P (see Fig. 8).

Consider the $\gamma_i/\bar{\gamma}_i$ calls (*). The corresponding $\gamma/\bar{\gamma}$ calls in

the semantic map 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 $\gamma_i(Q_r)$ is finite, or there is a true Q'_s for which $\bar{\gamma}_i(Q'_s)$ 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 1. If Γ_i is true then $\bar{\gamma}_i(P)$ for any interpretable P is finite only if P is false.

Indeed, should P be true, we would have a situation where a true proposition (Γ_i and P) produces a false atomic proposition.

Corollary 2. Every interpretable proposition is context-safe.

Corollary 3. If Γ_i is true, the formal system which takes Γ_i as its knowledge is consistent.

(Because, if both $\gamma_i(P)$ and $\bar{\gamma}_i(P)$ are finite then it violates either Correctness theorem or Corollary 1).

Theorem. If P is a false proposition then $\bar{\gamma}(P)!$.

Proof. We use induction on the structure of the interpretable proposition P.

Base. If P is $A!$ with a mechanical A, and it is false, then $A?$ is true. We can add $A?$ to the current knowledge, thereby making P contradictory and $\bar{\gamma}(P)$ finite. Thus there exists such a (true) knowledge Γ_i that $\bar{\gamma}_i(P)$ is finite. By the definition of dynamic interpretation, this means that $\bar{\gamma}(P)$ is finite. If P is $A?$, we add $A!$ to human knowledge.

Induction on generation. If P is a false generator, it produces at least one false proposition; let it be Q. By the induction hypothesis, $\bar{\gamma}(Q)$ is finite. Therefore, $\bar{\gamma}(P)$ is finite.

Induction on cognitive function calls. Let P be $A!$ and false. Suppose first that the search A has only one branch. Since $A!$ is false, at least one of the following three situations must take place:

(a) The search A' produced from A by 'short-circuiting' (that is replacing by the empty expression) all cognitive function calls is infinite.

(b) A' is finite, and $\gamma(Q)!$ with some false Q appears in A.

(c) A' is finite, and $\bar{\gamma}(Q')!$ with some true Q' appears in A.

Upon examination, case (c) is impossible. If Q' is true, then its addition to human knowledge cannot lead to contradiction.

In case (a) we add to human knowledge the proposition

if $A!$ then $A'!$

which is, obviously, true. Then we add $A'?$, which also is true. Now $A!$ produces a contradiction, namely $A'!$ and $A'?$.

In case (b) we add to human knowledge:

if $A!$ then Q

By the induction hypothesis, the addition of Q leads to a contradiction. Therefore, the addition of $A!$ also leads to a contradiction.

If the search A has more than one branch and is finite, then there must be at least one branch that is finite. Applying the reasoning above to this branch we can prove that $A!$ leads to a contradiction. Since the statement that A is finite (which, in addition, is false) does not give us any indication which of the branches is thought to be finite, we cannot indicate specifically what true statements must be added to human knowledge to ensure the derivation of a contradiction from $A!$. But the reasoning that if A is finite then there must be a finite branch, with the subsequent invoking of the above proof that the existence of such a branch is contradictory -- is intuitively beyond doubt and a part of classical logic. Therefore, it can be formalized as part of human knowledge in our theory; then a contradiction can be formally derived from $A!$, that is $\bar{\gamma}(P)$ is, in dynamic interpretation, finite.

▽

So, if P is false then $\bar{\gamma}(P)!$ is true. On the other hand, if $\bar{\gamma}(P)!$ is true then P must be false: if it were true it could not produce a contradiction. The statements 'P is false' and 'P leads to a contradiction' are equivalent. A pair of the form P and $\bar{\gamma}(P)!$ will be considered now as a formal contradiction, like $A!$ and $A?$. Indeed, if the addition of a proposition, say Q, produces both P and $\bar{\gamma}(P)!$, then one of the two is false, which means Q is false.

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. Orlov's ideas can probably be used in trying to expand the present theory aiming at description of subatomic phenomena.

CHAPTER 5

Logic

1. First-order theories

Let us sum up where we are. We defined mathematical propositions as generators of predictions. We found that in order to explicate in this way the most usual logical constructs, such as existential quantification and material implication, it is necessary to introduce into the theory a formalization of the process of *getting the knowledge* that a given proposition *P* is true. We denoted this process by $\gamma(P)$ and defined it as a logical inference of *P* from a proposition Γ, which represents the sum total of the knowledge accumulated up to date by humanity. We denoted the process of establishing that *P* is false by $\bar{\gamma}(P)$; it is the process of finding a contradiction in the logical conjunction of Γ and *P*. We translated the language of mathematical logic in terms of our theory.

Naturally, Γ is not a fixed expression, but it changes as time passes. Unlike mechanical processes, which we define using the Refal machine or any other linguistic machine, human knowledge is a *real-time* process; we cannot reproduce all of its stages in their sequence, as we can do with mechanical *model-time* processes; we can only know what its present stage is.

We defined the concept of truth as follows: a prediction *A*! is true if the process *A* is finite; a general proposition *P* is true if all the predictions it hierarchically produces are true. We recognized then the following fact, which is of the utmost importance: since human knowledge enters the meaning of propositions (the set of predictions produced and the processes referred to by predicttions), we cannot assume automatically that a proposition that is true today will be true tomorrow. We gave an example where adding a true proposition to a true knowledge results in a false knowledge; it happenes because by adding a new proposition to Γ we change the meaning of Γ and other propositions, and some of them may become false. We found a way out in exploiting the intuitive idea of *objectivity*, according to which

the cognitive processes τ and $\bar{\tau}$ only register the truth values of propositions, which 'are there' anyway. We defined a procedure of objective interpretation of a proposition, and agreed to accept as meaningful only those propositions for which this procedure leads to a definite truth value. However, the naive notion that some 'real' truth values 'are there' attached somehow to processes is not acceptable for us as part of our formalism; we have shown that *the dynamic interpretation* of references to τ is equivalent to the intuitive objective interpretation.

The procedure for establishing the interpretability of a proposition is by no means an algorithm with a guaranteed end. Interpretable propositions form a hierarchy which is constructed step by step, starting with the propositions that never refer to cognitive functions. At each step proofs are required, which rely on our current knowledge. To be valid, this knowledge must use only interpretable proposition; their interpretability, therefore, must have been established *prior to* their use for establishing interpretability of other propositions. The hierarchy of interpretable propositions which must be constructed in order to validate a given proposition as interpretable P^n includes not only those propositions P_1^{n-1}, P_2^{n-1}, etc., in terms of which P^n is interpreted, but also the propositions which are necessary to prove that P^n really *is* interpretable if P_1^{n-1}, P_2^{n-1}, etc. are interpretable. Interpretability and knowledge are inseparable.

We proceed now to examine in greater detail the structure of a mathematical theory. First of all, every theory has certain *objects* to deal with, and certain *primary predicates* applicable to objects. Then there are certain *logical forms* to make composite predicates from primary ones: connectives and quantifiers. All these are usually defined purely syntactically, without any indication as to what the meaning of the predicates and logical forms is. It is entirely left to the stage of application. In our approach, the logical forms and the primitive predicates of a mathematical theory are *parametrized processes*. The logical forms are *machines* defined in Refal, namely: imp, con, if, and, or, all, sch. The primitive predicates may be defined by machines, or left undefined (see Section 3.2 about cybernetic and axiomatic theories); in the latter case their definition is left for applications. All propositions in our approach have a definite *meaning* expressed in terms of processes, as we have just summed up above.

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. The origin of the terms is this. When you have some objects, you can create a language and a theory to treat these objects. It is a theory of the first order. Then you take the propositions of this theory as the objects for a new theory, which is thereby a theory of the second order, etc. In practice, mathematics uses only one theory, but it is of the infinite order. It is set theory. 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 this chapter we consider the common features of the first-order theories, which amounts to considering logic. *First-order logic*, they often say.

A first-order theory is a real-time process each stage of which is a formal system as defined in Chapter 4. According to that definition, a formal system consists of two parts: a number of machines, and a knowledge Γ. The mechanical part of any first-order theory includes all the logical machines listed above. Now we are about to create the knowledge part.

In accordance with the strategy formulated in Chapter 4, we go ahead by steps, starting with an empty knowledge and adding propositions which we have proved to be true in some way which our intuition accepts without doubt. Our purpose is to reconstitute all the usual apparatus of formal logic, thereby proving that logic is true and consistent. As we go on, we shall consider one or another proposition P with an aim to justify its use in logic, if possible. We have two ways to do it. First, we can show that with the knowledge Γ we have proved true up to the moment, P is deducible in our formal system, i.e. implied by Γ. We shall represent this proposition by the formula

$$\Gamma \rightarrow P$$

An equivalent form of it would be $\gamma(P)!$. Second, if P is not implied by Γ we can prove intuitively that it is true and include P into Γ, i.e. change the current Γ to $\underline{\text{and}}(\Gamma,P)$. This is not merely a proposition, but an act in real time, which we called a *statement* in Chapter 4. We shall represent this statement by the formula

$$\Gamma \leftarrow P$$

We shall refer to the set of all propositions produced hierarchically by Γ as its *scope*. If a proposition P is implied by Γ, directly or hierarchiclly, its scope is, of course, a subset of the scope of Γ (see Fig. 5.1). If Γ is true then every P implied by it is also true, which is a trivial case. But a proposition, such as P' in Fig.5.1, may have a scope which is a subset of the scope of Γ without being ever produced (implied directly or hierarchically) by Γ. We say that P' is *justified* by Γ. It is true if Γ is true. Instead of proving P' intuitively, we can prove that it is justified by Γ and add it by the corresponding statement. It should be noted that the addition of a proposition P' justified by Γ may expand the scope of $\underline{\text{and}}(\Gamma,P')$ beyond the union of the scopes of Γ and P'. This situation is shown schematically in Fig. 5.1. An example when it takes place follows:

$$\Gamma = \underline{\text{and}}(A!,\underline{\text{and}}(B?,\underline{\text{if}}\gamma(\underline{\text{and}}(A!,B?))!\ \underline{\text{then}}C!)$$
$$P' = \underline{\text{and}}(A!,B?)$$

The scope of Γ consists here of the following five propositions:

(1) Γ itself
(2) $A!$
(3) $\underline{\text{and}}(B?,\underline{\text{if}}\gamma(\underline{\text{and}}(A!,B?))!\ \underline{\text{then}}C!)$
(4) $B?$
(5) $\underline{\text{if}}\gamma(\underline{\text{and}}(A!,B?))!\underline{\text{then}}C!$

The proposition $\underline{\text{and}}(A!,B?)$ is not among them, therefore the $\underline{\text{if}}$-proposition (5) produces nothing. The proposition P' produces two atomic propositions $A!$ and $B?$, which are also produced by Γ. Therefore, it is justified by Γ. When we add P' to Γ, the resulting system produces $\underline{\text{and}}(A!,B?)$, and as an additional result

produces the atomic proposition C!

Let <obj> (just obj in the free format) be the generator of all objects of a first-order theory. Then all quantifications will take place over this set, and we can omit references to it the way it is usually done in formal logic. Thus instead of

$$\underline{\text{all}}(x \epsilon \underline{\text{obj}} : P(x))$$

we shall write

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

and the same for sch.

When constructing a theory we want to be sure that all propositions of it are *interpretable*. With first-order theories this problem is solved very easily. One can verify that all the logical machines we defined are such that when their arguments are interpretable, the resulting processes are also interpretable. When primitive predicates are defined by machines, they would not include references to cognitive functions, and therefore initiate interpretable processes. Since the whole set of possible propositions is produced by substitutions, there is no way an uninterpretable proposition can appear in a cybernetic theory. The same will be true for an axiomatic theory if the axioms are all interpretable. It would be strange if they were not, because axioms are created as a formalization of the properties of natural processes, which are thought of in objective terms and leave no place for cognitive processes.

It follows from the preceding that not only the set of all objects but also the set of all processes and propositions in a first-order theory is produced by mechanical generators (are recursively enumerable). Let pser be the generator of all processes-searches, pgen the generator of all processes-generators, and prop the generator of all propositions. This allows us to quantify variables over these sets in order to express very general propositions of logic. Suppose we have a propositional form P(p) which depends on the propositional variable p. Then we can form, and possibly add to Γ, a single proposition

$$\underline{\text{all}}(p \epsilon \underline{\text{prop}} : \mathbf{P}(p))$$

which will be equivalent to the set of propositions P(*P*) where *P* stands for an arbitrary proposition, i.e. is used as a metasymbol. Since this second form is more readily understandable, we are going to prefer it to the quantified form, but one should be aware that such propositional forms stand for single quantified propositions, and there is no need in our theory to introduce anything like axiom schemata of mathematical logic. Even most general logical laws are expressed in our theory by single linguistic objects.

To finish up with the preliminaries we only have to sum up the results of Chapter 3 in the form of a table translating the usual logical notation into the corresponding propositions of our theory. The translation of a conventional logical proposition *P* will be denoted by [*P*]. Primitive predicates are translated according to their meaning if we construct a cybernetic theory, or as undefined parametrized processes, for which only the name must be chosen, if we construct an axiomatic theory. For composite propositions the translation rules are:

[~*P*]	=	~([*P*])!
[*P*&*Q*]	=	and([*P*],[*Q*])
[*P*∨*Q*]	=	or(∨([*P*],∨([*Q*]))!
[*P*→*Q*]	=	if∨([*P*])!then[*Q*]
[(*Ax*)*P*(*x*)]	=	all(*x*:[*P*(*x*)])
[(*Ex*)*P*(*x*)]	=	sch(*x*:∨([*P*(*x*)]))!

2. Definition and verification

One of the most certain parts of our knowledge is what we know *by definition*, because we set it to be that way. The machines we use are defined by sentences in the program field of the Refal machine. Let us see how the knowledge contained in these definitions can be formalized.

Consider the first sentence of a function definition, and suppose its left side *L*, and therefore also its right side *R*, includes no free variables. Then one step of the Refal machine will transform *L* into *R*. This can be expressed as the proposition:

(1) ↑<=(<step↑*L*>)(↑*R*)>!

which encapsulates the whole knowledge contained in the original sentence. We shall say that the expression E_x *immediately entails* E_y if one step of the Refal machine transforms E_x into E_y. The proposition (1) states that L immediately entails E_y.

The predictions of mathematics state the finiteness of processes without indicating how many steps it takes for a process to stop. Pure mathematics, unlike computer science, does not count computation steps. Therefore, the transitive closure of immediate entailment will be more useful for mathematical purposes. We say that E_x *entails* E_y if E_x is transformed by the Refal machine into E_y in any number, but at least one, steps. It is easy to define the function which tests this relation:

```
<ent(e_x)→e_y>  →  <entl(<step e_x>)→e_y>
<entl(e_x)e_x>  →  e_x
<entl e_z>      →  <ent e_z>
```

The process $\langle\underline{\text{ent}}(\uparrow E_x)\rightarrow\uparrow E_y\rangle$ ends if and only if E_x entails E_y. Instead of (1) we can now use this proposition:

(2) $$\uparrow\langle\underline{\text{ent}}(\uparrow L)\rightarrow\uparrow R\rangle!$$

If L includes free variables, they can be universally quantified in the propositions (1) and (2); if they are left free, the quantification is assumed when we deal with such propositions. If the sentence $L \rightarrow R$ is not the first in the group of sentences defining a function, then we cannot simply transform it into (1) or (2), because this sentence will be used only if all the preceding sentences of the definition are unapplicable. This means that the free variables in L cannot be universally quantified, but exceptions must be made for those possible cases when this sentence will not be really used. For instance, if the function fun is defined by two sentences

```
<fun Ae_z>   →  B
<fun s_x e_z>  →  s_x
```

then in the proposition representing the second sentence s_x must be quantified over all symbols except A.

It is possible to represent all the information contained in the definitions of a theory in the form of propositions of en-

tailment. Also, as we shall see in a moment, all individual propositions about entailment can be produced in a simpler way, using only one very powerful statement, to the discussion of which we now proceed.

This statement is *the verification principle*:

$\Gamma \leftarrow$ all($x \in$ pser : if x! then x!)

In the form with metavariables:

$\Gamma \leftarrow$ if A! then A!

where A is an arbitrary search.

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

finite(A) $\rightarrow$ finite(A)

which is of little, if any, use. In our system, however, the verification principle is far from being trivial, and not an axiom but a theorem. Let us see what is the full scope of propositions it hierarchically produces. We add to our knowledge Γ a proposition which produces the propositions

(3) if A! then A!

with all possible searches A! expressible in the theory. When Γ starts working, every proposition (3) 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! for which A is defined and finite will be produced by Γ. The verification principle formalizes the fact that the finiteness of a search which is defined mechanically can be directly verified, at least in principle. The generator (3) produces all propositions that can be proved true through verification. It is an intuitive theorem of our metatheory, which we have just proven by establishing that the verification principle produces only true propositions.

Let E_x entail E_y. Then the search ent(E_x,E_y) is defined and finite, therefore ent(E_x,E_y)! is produced by the verification principle. Thus the verification principle alone provides for all

individual relationships of entailment which can be derived from the definitions in the memory field. (But it does not provide for quantified propositions of entailment like (1). They are *justified* by the verification principle, but not produced by it).

Definition. Two parametrized searches S_1 and S_2 are *functionally equivalent*: $S_1 \equiv S_2$, if one of the following two situations takes place with any values of free variables: (1) both searches are infinite, or (2) both searches are finite and their results (finite stages) are identical.▽

The following general statements can be made, which are obviously true:

$$\Gamma \rightarrow \underline{\text{if}}\ \underline{\text{ent}}(S_1 \rightarrow S_2)!\underline{\text{then}}\ S_1 \equiv S_2$$
$$\Gamma \rightarrow \underline{\text{if}}\ \underline{\text{ent}}(S_2 \rightarrow S_1)!\underline{\text{then}}\ S_1 \equiv S_2$$
$$\Gamma \rightarrow \underline{\text{if}}\ \neg(S_1 \equiv S_2)!\underline{\text{then}}\ S_2 \equiv S_1$$

Here S_1 and S_2 are arbitrary parametrized searches.

If the machinery of a formal system includes the functions stepu and actu, then the introduction of new machines by adding their definitions to the memory field adds really nothing to the power of the existing machinery. Indeed, suppose we define a certain function, F, by the list of sentences S. The expression

$$\langle \bar{\mu} \langle \underline{\text{actu}}(\underline{\text{and}}(D, \uparrow S))\ *(F\langle \mu e_x \rangle) \rangle \rangle$$

where D is the list of definitions currently in the memory, is functionally equivalent to $\langle F e_x \rangle$. Since we do not care in mathematics how long it takes to compute something, we can use the former wherever we use the latter. The definition of new functions by adding new text to the memory field becomes just a matter of convenience, a notation. Whenever we say 'define a function by such and such sentences', we also could say 'consider the function actu with such and such first argument'. Since we introduce the functions stepu and actu at the early stages of our theory, we shall assume that all the definitions made later become automatically known to the formal system.

3. Basic logical principles. Intuitionism

In this section we construct that part of logic which does not depend on how we treat human knowledge Γ when determining the

truth values of propositions. As mentioned before, it can be done in two ways. Γ can be considered as fixed: a definite, though perhaps very big, expression. It is not required that Γ is actually written down; for practical purposes of making proofs we draw, as always, on our intuition, and when we prove something we consider it as included in Γ. When reasoning theoretically in this approach, we do not use any specific features of Γ, but only the fact that it is *definite*. This is not to deny that Γ changes in real time. We simply refuse to speculate on what can happen to Γ in the future, and identify the truth of any proposition with its deducibility from the Γ of today. This is the *static* interpretation of Γ. The alternative *dynamic* interpretation claims that even those propositions that can neither be proved nor refuted today must be true or false; it does speculate on future, boldly tying up the concept of truth with the future of the process Γ. A major part of logic does not depend on the interpretation we keep to. It is common to the intuitionist and the classical logic.

So we start constructing logic from scratch. First we create the necessary machinery: put in the memory of the Refal machine the definitions of all logical machines discusses above. At this stage the knowledge Γ of out theory is still empty. Let us take the verification principle as our first statement (see the preceding section). Now Γ produces all propositions of the form:

(1) $\underline{\text{if}}A!\underline{\text{then}}A!$

where A is a search.

For an arbitrary P, substitute $\gamma(P)$ for A in (1):

(2) $\underline{\text{if}}\gamma(P)!\underline{\text{then}}\gamma(P)!$

All possible propositions (2) are produced in Γ as a subset of the scope of the verification principle. If P is produced by Γ, then $\gamma(P)$ is finite. Then (2) produces $\gamma(P)!$. We can now substitute $\gamma(P)!$ for P in (2), and so on. We conclude that if P is produced in a formal system Γ (this is for short instead of 'a formal system with the knowledge Γ'), then the propositions $\gamma(P)!$, $\gamma(\gamma(P)!)!$, etc. are also produced in it. In words: If Γ knows P, then it also knows that it knows P, and it knows that it knows that it knows P, etc.

Suppose now that $\Gamma\rightarrow\gamma(P)!$. Since Γ is true, the fact that $\gamma(P)!$ is produced by Γ means that it is actually true, i.e. that $\gamma(P)$ is finite. But its finiteness means that $\Gamma\rightarrow P$. Combining this result with the preceding one yields the following

Theorem. Propositions P and $\gamma(P)!$ are equivalent in Γ,

$$P\equiv\gamma(P)!$$

meaning that whenever one is produced by Γ the other is produced too.

We can now easily prove theorems corresponding to well-known logical identities. For instance,

Theorem. Implication is transitive:

$$\Gamma\leftarrow \quad \underline{\text{if}}\,\gamma(\underline{\text{if}}\,\gamma(P)!\,\underline{\text{then}}\ Q)!\,\underline{\text{then}}\ \underline{\text{if}}\,\gamma(\underline{\text{if}}\,\gamma(Q)!\,\underline{\text{then}}\ R)!\,\underline{\text{then}}\ \underline{\text{if}}\,\gamma(P)!\,\underline{\text{then}}\ R$$

It can be made easier to review in the form of a derivation rule:

#1 $\Gamma\leftarrow \underline{\text{if}}\,\gamma(P)!\,\underline{\text{then}}Q$

#2 $\Gamma\leftarrow \underline{\text{if}}\,\gamma(Q)!\,\underline{\text{then}}R$

#3 $\Gamma\rightarrow \underline{\text{if}}\,\gamma(P)!\,\underline{\text{then}}R$

Proof. The proposition in (#3) produces R whenever Γ produces P. But if Γ produces P then, according to (#1) it also produces Q, and if it produces Q then, according to (#2), it produces also R. Therefore (#3) produces R only in the case when Γ already produces it anyway. The proposition in (#3) is *justified* by Γ, therefore the statement (#3) is *correct*. Note that although the proposition in (#3) is justified, it is not produced by Γ, this is why (#3) is a *statement* $\Gamma\leftarrow$, not a *proposition* $\Gamma\rightarrow$. After this statement our implication becomes transitive, but not before it. The proof of the theorem is the proof that the statement leaves the current knowledge true if it was true before. As we mentioned before, the theorems we prove while constructing a formal system either show that a desired feature is there already, or justify the next construction step, which is made in order to have this feature. This theorem is of the latter type.▽

To show the equvalency of our construction to a formal system already known we must interpret in our terms the axioms and the derivation rules of that formal system. We shall do it for the part of mathematical logic which is common to classical and intuitionistic logic, and this part is coextentional with intuitionist logic, because every proposition provable in intuitionist logic is also provable in classical logic. The inverse is not true; classical logic can be constructed from intuitionist logic by adding one more axiom: the law of the excluded middle or its equivalent.

The intuitionist propositional calculus can be based, according to *Heyting (1956)*, on eleven axioms (*H1-H11*),which we prove below as theorems in our system.

Theorem H1: P → P&P

From the definition of the and machine it is obvious that and(*P*,*P*) generates only *P* (though twice). The statement

Γ ← if$_{\gamma}$(*P*)then and(*P*,*P*)

produces *P* only it is already produced by Γ. Thus it is justified.

Theorem H2: P&Q → Q&P

If and(*P*,Q) is produced, then *P* and Q are also produced, and therefore and(Q,*P*) is justified:

Γ ← if and(*P*,Q)!then and(Q,*P*)

Theorem H3: (P→Q) → (P&R → Q&R)

Let

(1) Γ→ if$_{\gamma}$(*P*)!thenQ

and

(2) Γ→ and(*P*,*R*)

From (2) we see that Γ produces *P*, therefore (1) will produce Q. It also produces *R*, because of (2). This proves that H3 is *justi-*

fied.

Theorem H4: $(P \rightarrow Q)\&(Q \rightarrow R) \rightarrow (P \rightarrow R)$

This theorem follows readily from the definition of the and machine and the transitivity of implication which we proved above.

Theorem H5: $Q \rightarrow (P \rightarrow Q)$

The statement

$$\Gamma \leftarrow \underline{\text{if}}_{\gamma}(Q)!\underline{\text{then}}\ \underline{\text{if}}_{\gamma}(P)\underline{\text{then}}Q$$

does nothing more than add $\underline{\text{if}}_{\gamma}(P)!\underline{\text{then}}Q$ to Γ if Q is produced by Γ. Since this non-atomic statement can produce only Q again, it is justified.

Theorem H6: $P\&(P \rightarrow Q) \rightarrow Q$

The statement

$$\Gamma \leftarrow \underline{\text{if}}_{\gamma}(\underline{\text{and}}(P,\underline{\text{if}}_{\gamma}(P)!\underline{\text{then}}Q))!\underline{\text{then}}Q$$

produces Q only if P and $\underline{\text{if}}_{\gamma}(P)!\underline{\text{then}}Q$ are produced. But in that case Q is produced anyway.

Theorem H7: $P \rightarrow P \vee Q$

Consider the statement:

$$\Gamma \leftarrow \underline{\text{if}}A!\underline{\text{then}}\ \underline{\text{or}}(A,B)!$$

where A and B are any processes. If A is finite it produces the prediction that $\underline{\text{or}}(A,B)$ is finite. This is a true prediction, as one can immediately see from the definition of the or machine, which runs A and B in parallel until either of them stop.

Theorem H8: $P \vee Q \rightarrow Q \vee P$

This theorem also immediately follows from the definition of the or machine.

Theorem H9: $(P \rightarrow R)\&(Q \rightarrow R) \rightarrow (P\&Q \rightarrow R)$

Suppose that the antecedent of this implication is true, i.e.

$\Gamma \leftarrow$ $\underline{\text{if}}\gamma(P)!\underline{\text{then}}R$
$\Gamma \leftarrow$ $\underline{\text{if}}\gamma(Q)!\underline{\text{then}}R$

Then the statement

$\Gamma \leftarrow$ $\underline{\text{if}}\gamma(\gamma(P),\gamma(Q))!\underline{\text{then}}R$

is justified because it produces R only if at least one of the processes $\gamma(P)$ and $\gamma(Q)$ is finite, but in that case R is already produced by Γ.

Theorem H10: $\sim P \rightarrow (P \rightarrow Q)$

This corresponds to the statement:

$\Gamma \leftarrow$ $\underline{\text{if}}\bar{\gamma}(P)!\underline{\text{then}}$ $\underline{\text{if}}\gamma(P)!\underline{\text{then}}Q$

which will produce something only if both $\gamma(P)$ and $\bar{\gamma}(P)$ are finite. But this is possible only if Γ is contradictory. Indeed, $\bar{\gamma}(P)!$ means $\underline{\text{con}}(\Gamma\&P)!$. Since $\gamma(P)!$, the proposition P is produced by Γ, therefore the $\underline{\text{con}}$ machine will find a contradiction with Γ alone on the input: $\underline{\text{con}}(\Gamma)!$. But this is imposiible because our system was consistent in the beginning, when it was empty, and has remained consistent after every statement. Therefore **H10** represents an *empty* proposition, and as such is justified.

Theorem H11: $(P \rightarrow Q)\&(P \rightarrow \sim Q) \rightarrow \sim P$

Suppose that the antecedent is true. Then

$\Gamma \leftarrow$ $\underline{\text{if}}\gamma(P)!\underline{\text{then}}Q$
$\Gamma \leftarrow$ $\underline{\text{if}}\gamma(P)!\underline{\text{then}}\bar{\gamma}(Q)!$

Form Γ' as $\underline{\text{and}}(\Gamma,P)$. Both $\gamma(Q)!$ and $\bar{\gamma}(Q)!$ are produced in Γ', which means that it is contradictory: $\underline{\text{con}}(\underline{\text{and}}(\Gamma,P))!$. We denote this as $\bar{\gamma}(P)!$, so **H11** is justified.

When all the statements referred to in the proofs of H1 to H11 are made, we have a formal system in which every axiom of the intuitionist propositional calculus is produced. We want to prove now that every theorem of that calculus is produced also. To

achieve this goal it suffices to show that if a proposition of logic, say *P*, can be obtained by applying one of the *inference rules* of the logic to some propositions already proven (including, of course, the axioms in that number), then *P* is also produced in our formal system Γ. There are two inference rules in the propositional calculus. The *Substitution* rule allows one to substitute any proposition for a propositional variable. In our sytem it is taken care of by the definition of universal quantification. The all machine will produce all those propositions which can be proved in logic by applying the substitution rule. The *Modus Ponens* rule of logic declares that if *P* and *P*→*Q* are proved then Q becomes proved. It is taken care of by the definition of the if machine in our system. Indeed,

if	Γ→	*P*
and	Γ→	if∨(*P*)!thenQ
then	Γ→	Q

To construct the predicate calculus, we must add to the axioms of the propositional calculus two axioms: **AP1** and **AP2**, and two rules of inference: **IP1** and **IP2**. They are also sufficient for the predicate calculus in classical logic. We prove them as theorems.

Theorem AP1: $(Ax)P(x) \rightarrow P(r)$

where $P(x)$ is a proposition depending on the free variable x, and r is an object.

This axiom, like the substitution axiom, is taken care of by our definition of universal quantification, the difference being that the quantification is over the set of all legitimate terms, not propositions.

Theorem AP2: $P(r) \rightarrow (Ex)P(x)$

To cover this axiom, we make the statement:

Γ ← if*A*(*R*)!then sch(*x*:*A*(*x*))!

where *A*(x) is a metavariable for any process parametrized on *x*, and *R* a metavariable for an object. It is justified by the definition of the sch machine. If there is such an object *R* that *A*(*R*) is finite, then the sch, which executes all the processes *A*(*x*) in

parallel, is bound to find R, if nothing else, and stop. The substitution of $\gamma(P)$ for A proves the theorem.

Now to the inference rules.

Theorem IP1: If $Q \rightarrow P(x)$ is true, and x does not appear freely in Q, then $Q \rightarrow (Ax)P(x)$ is also true.

This rule of inference, as well as the next one, deals with propositions which include free variables (propositional forms). A free e-variable in a Refal expression stands for an arbitrary object expression. It may happen that when we are (or rather the Refal machine is) processing an expression, some subexpressions of that expression are not actually analyzed or changed: they are manipulated as some unknown wholes. Such subexpressions could be replaced by free variables during the processing, and the Refal machine would not notice that they were free variables and not regular (object) expressions. For instance, with the function

$$\langle \underline{\text{cdr}}\ s_x e_y \rangle \rightarrow e_y$$

the expression <cdr AB(1234)> is concretized as B(1234). We also can say that <cdr Ae_x> is concretized as e_x, and <cdr $AB(e_x)$> is concretized as $B(e_x)$, etc. But we can say nothing about <cdr e_x> or <cdr e_x AB>. Whatever the contents of the memory of the Refal machine, if an expression including free variables, say E, passed a certain processing successfully, then any expression resulting from the substitution of arbitrary object expressions of corresponding syntax types in place of free variables in E, will also successfuly go through the same processing. In other words, free variables in propositions must be interpreted as universally quantified.

If the premise of the rule is true, i.e.

(1) $\underline{\text{if}}\gamma(Q)!\underline{\text{then}}P(x)$

is produced by Γ, then every proposition resulting from substitution of any object for x into (1) is true. Since Q does not depend on x, we can first, and only once, check that $\gamma(Q)$ is finite, and then produce all the propositions $P(x)$. This justifies adding (1) to Q.

Theorem IP2: If $P(x) \rightarrow Q$ is true, and Q does not contain x

freely, then $(Ex)P(x) \rightarrow Q$ is also true.

If the premise is true, then every proposition

(2) $\underline{\text{if}}\gamma(P(X))!\underline{\text{then}}Q$

with an arbitrary object X is true. Note that Q is the same in all these propositions because it does not depend on x. Suppose now that $(Ex)P(x)$ is true. This means that there is at least one object X_0 for which $\gamma(P(X_0))$ is finite. Substituting X_0 for X in (2), we get a true proposition which produces Q. Therefore, Q is true. Thus, if $(Ex)P(x)$ is true then Q is true. This proves the consequence of the theorem, and we can make the corresponding statement.

This completes the construction of a formal system known as intuitionist logic. That it is intuitionist can be seen not from what is included, but from what is left out. The different concepts of mathematical truth embraced by intuitionist logic and classical logic manifest themselves in the way these theories treat *negation*. For classical logic, the truth values of propositions are sort of *tags*, 'objectively' attached to them, and there are exactly two truth values: true and false. To state a proposition is equivalent to stating that it is true. To negate it means to state that it is false. To negate the negation of a proposition is to state that it is true. This is *the double negation law*:

$$\sim\sim P \rightarrow P$$

Every statement is either true or false. There can be no third possibility. This principle is known as *the law of the excluded third* (or *middle*):

$$P \vee \sim P$$

Intuitionism regards a proposition as the expression of some mathematical reasoning, or proof, which is trusted by our intuition. To state a proposition is to state that you have a proof of it. To negate a proposition is to state that you have a proof that it leads to contradiction. If this view is taken, then there are no immediate reasons to believe in the law of the excluded third. It is possible that you can neither prove nor disprove a proposition -- this is the third logical possibility. Also, the

law of double negation loses its grounding: if there is no proof that P is contradictory, it does not yet follow that P is true.

Our theory provides a convenient means to understand intuitionism and its relation to classical logic.

In our formalism, the double negation of a proposition P is

$$\bar{\gamma}(\bar{\gamma}(P)!)!$$

It reads: it is contradictory to Γ that P is found contradictory to Γ. It is obviously not the same as to say that P is true: $\Gamma(P)!$. The disjunction in the law of the excluded middle is:

$$\underline{\text{or}}(\gamma(P),\bar{\gamma}(P))!$$

It states that every proposition P is either implied by Γ or contradicts to it. It is far from being obvious. In fact, if we accept the static interpretation of Γ, i.e. assume that Γ is a definite expression, then it becomes definitely false. For we know from Goedel's theorem that for every formal system there is a proposition which can be neither proved nor refuted in it. Therefore, if we accept the static interpretation of Γ we come inevitably to intuitionist logic.

4. Goedel's theorem.

Of the two types of atomic propositions, on which, in the last analysis, the whole edifice of mathematics is built, the first type, the prediction *A*! is directly verifiable, at least in principle. The second type, the infinity model *A*?, is not; in order to establish that *A* is infinite, we have to rely on some proof based on some knowledge, i.e. a conjunction of truths. Is it possible to find a knowledge such that it would enable us to prove the infinity of any mechanincal process *A*, if it in fact is infinite? Note that to discover such a knowledge would mean, in a sense, to outsmart nature. For the impossibility of reducing the infinite to the finite is deep in the nature of things. By definition, you cannot examine all the stages of an infinite process and come to an end. And if you cannot examine all its stages how can you know that a final stage will never be reached? One, and probably the only, way is to analyze the conctruction of the machine that originates the process, and demonstrate that none of the stages which could terminate the process can actually occur. The simplest example would be a process where an active stage reproduces itself and only itself at each next step. Yes, you can construct a lot of processes of this kind, a big library of prototypes or patterns which originate infinite processes, more and more complex. If somebody gives you a definition of a process, you may be able to discern in it some patterns familiar from your library, like the simple circularity. Then you will know that the process is infinite. This is exactly what the mathematicians are doing. But it should be clear that the number of possible patterns of infinity is infinite. We build the library starting from the end of the finite, and no matter how big the library is, it will never reach the end of the infinite. The general problem demands that we start from the other end: an arbitrary infinite process is given, and one must check that it is really infinite. Our library of patterns does not bring us closer to the solution of this problem. Looking at the problem from this angle, we start to realize that it would be a miracle should a device be found which allowed us to solve it, because it would have been a device which 'outsmarts' infinity, essentially liquidates it. The famous Goedel's theorem of incompleteness tells us that there will be no such miracle.

The idea behind Goedel's theorem, when put in terms of our theory, is this. Suppose we can construct a process *G* which is

proving (trying to prove) in a certain theory its own infinity. Then the following two statements must be true in the assumption that the theory is true:

(1) The process G is infinite. Indeed, if it is finite, then its infinity in out theory is proven, and since the theory is true, G must be infinite, which is contradictory and,therefore, impossible.

(2) It is impossible in our theory to prove that G is infinite. Indeed, if the proof of the infiniteness of G is finite then G, which is exactly the process of proving that, is finite, which, as we have just established in (1), is impossible.

Therefore, G? is a proposition which is true but not provable in the theory. If the process of proving its own infiniteness is expressible in a theory, and the theory is true, it is incomplete.

To apply this idea to formal arithmetic, Goedel had to map proofs on numbers, a procedure that became known as *Goedelization*. In our theory, the very general nature of objects and processes we deal with, plus the transformation of metacoding, makes unnecessary any additional Goedelization. In fact, our metacode transformation is a generalization of Goedelization: it transforms processes, in particular the processes of proving, into objects, in particular, the objects of discourse. Our general term for this is: metasystem transition. The important feature of our formalism is that it is designed not for theoretical purposes only, like showing the non-existence of this or that algorithm, but for the actual creation of intelligent computer systems. The Goedelization procedure actually used by its author is completely unfit for practical purposes.

The most straightforward definition of the Goedel process G is:

(1) $\langle G \rangle \rightarrow \langle \gamma^*(G)? \rangle$

It is a legitimate definition of a process in Refal. Should *(G)? also be a legitimate proposition in our theory, the theory would be ruined. Indeed, we saw that in our main, *dynamic* interpretation of Γ, which we expect to yield the explanation of classical logic and set theory, $\gamma(P)$ must be finite for every true P, i.e.

the theory -- by which we now mean not any formal system, but the metatheory, i.e. our whole theory of mathematics -- must be complete. But this contradicts the impossibility of proving the proposition G? (which is *(G)? in the strict notation), which has been established earlier. To repeat the reasoning with regard to (1), if <γ*(G)?> is finite, then by (1), <G> is also finite; but then the theory is false because *(G)? is validated by γ. Therefore, <γ*(G)> is infinite and <G> is infinite; but then the metatheory is not complete because γ fails to validate the true statement *(G)?.

This disaster does not happen to our theory because the process <G> as defined by (1) is *uninterpretable*, and so are all the propositions using it. According to the definition of interpretability, for the proposition *A*? to be interpretable *A* must not refer to cognitive processes. But *G* does. We can eliminate the question mark by redefining *G* so that it proves not its infinitness, but the falseness of its own finiteness:

(2) <G> → <$\bar{\gamma}$*(G)!>

This definition is equivalent to (1) and leads to all the same contradictions. But it is not interpretable either. To prove that <G> is interpretable we must *first* prove that the argument of $\bar{\gamma}$, i.e. *(G)! is interpretable; but to prove this we must *first* prove that <G> is interpretable. Since we cannot do it, the process is, according to point I.5 of the definition of interpretability, uninterpretable. It is not interpretable in the weak sense either, as one can check constructing its semantic map (see Fig 4.7). It is a typical example of infinite semantic recursion.

The concept of completeness, as it is currently used, is applied to theories which are identified with formal systems. We draw a difference between these two terms, so we must see what it changes with regard to the concept of completeness. Recall that by a formal system we mean, as usual, a purely mechanical device to produce true propositions. A theory is a real-time process each stage of which is a formal system. By the *scope* of a theory we mean the set of all processes which can be used in propositions of the theory. It may or may not depend on real time. The *knowledge* of a theory is the part of human knowledge pertaining to the scope of the theory. Total human knowledge is the sum

$$\Gamma = \Gamma^1 \& \Gamma^2 \& \Gamma^3 \& \ldots$$

of the knowledges r^i of different theories. In a particular case, a theory may stop in real time and reduce to a mechanical theory, a formal system. Finally, by the *metatheory* we mean our general theory of creating, justifying, and further developing theories

A theory is *complete* if for every proposition P of this theory either $\gamma(P)$ or $\bar{\gamma}(P)$ is finite:

(3) $$\underline{or}(\gamma(P),\bar{\gamma}(P))!$$

It is taken for granted that the theory is true (and therefore consistent). Then for every true P the process $\gamma(P)$ is finite and $\bar{\gamma}(P)$ infinite, and for every false P the process $\bar{\gamma}(P)$ is finite and $\gamma(P)$ infinite. There can be no third, or middle. Our statement of the completeness of a theory is identical to the statement that the law of the excluded middle holds for all legitimate propositions of the theory. It differs from the usual notion of completeness (i.e. that of a formal system) because of the involvement o real time; it deals not with the current state of a theory, but with its theoretical limit; the cognitive functions are metamechanical, not mechanical devices.

If, however, we consider a *stopped* theory, i.e. a formal system , then the cognitive functions γ and $\bar{\gamma}$ reduce themselves to some mechanical functions γ_i and $\bar{\gamma}_i$, and proposition (3) becomes:

(4) $$\underline{or}(\gamma_i(P),\bar{\gamma}_i(P))!$$

This corresponds exactly to completeness in the usual sense: for every legitimate proposition P, the theory gives a mechanical means to decide whether it is true or false.

Our general metatheory allows any processes that can be defined in Refal, and any propositions about these processes. To state the completeness of this theory is to state the law of the excluded middle.

But we can consider less general theories. The set of legitimate processes of the theory can be limited in one way or another, and all other processes declared undefined, i.e nonexistent as far as the theory is concerned. In such theories the criterion of completeness will be easier to meet.

Take, for example, the theory *T* in which only one process *A* is defined, and it is finite; and let the knowledge of the theory consist of one prediction *A!*, with the cognitive functions defined as in the general theory. Then the only propositions in the theory are *A!* and *A?*. This theory is true and complete. Goedel's proposition is not expressible in it.

Consider a mechanical theory, a formal system. A natural restriction on it would be the requirement to use only a certain number of machines defined beforehand. If the machines step and act are in that number, this is no significant restriction at all, because using them we can emulate any Refal definition. Since our general theory includes the definitions of these functions we do not care if we are limited to the predefined machines only. As we saw, if arbitrary Refal definitions are allowed in a theory, the Goedel statement is expressible through the definitions (1) or (2), where γ and $\bar{\gamma}$ are now mechanical functions. But the possibility of arbitrary Refal definitions is a rather strong requirement. We are going to abandon it and explore what kind of machinery is sufficient for the Goedel proposition to be expressible in a theory.

To include in consideration less powerful systems, we generalize the definition of what a formal system is. We do not require that any of the functions we introduced before are defined. There are only two functions which must be necessarily defined in every formal system, namely those which do the job of γ and $\bar{\gamma}$ in the general theory. We shall denote them g and $\bar{g}$, respectively, and call them the *proving machines* of the theory in order to stress their mechanical nature and not to confuse them with the cognitive functions of our general theory. The way g and $\bar{g}$ are defined is left open. But we still use Refal as the metalanguage to define processes, and we understand propositions as before.

To take an example, the simple system we discusses above can be completely defined by these sentences:

```
<γA!>  →
<γ̄A?>  →
```

and have no other processes defined at all. We left out even the definition of *A*, making our theory axiomatic. If a theory allows propositions about undefined processes known only through axioms,

the set of such processes must be additionally defined. In this case it consists of one process A which is known to be finite.

We shall now make two assumptions about the formal system, which we shall show to be sufficient for the proof of incompleteness.

(1) The proving machines of the theory are defined as specializations of a certain *class* of machines:

$$\langle g e_p \rangle \rightarrow \langle u(D) e_p \rangle$$
$$\langle \bar{g} e_p \rangle \rightarrow \langle \bar{u}(D') e_p \rangle$$

where u and $\bar{u}$ are certain universal functions with the formats $\langle u(e_d)e_p \rangle$ and $\langle \bar{u}(e_d)e_p \rangle$ characterizing a class of formal systems, and D is an expression which specifically defines the formal system in question.

This assumption is commonly justified. In the case of our general theory, u and $\bar{u}$ are imp and con (the latter slightly modified by including and), and the definition D is the knowledge of a specific formal system r_i. In usual formal logic, the machines u and $\bar{u}$ implement the inference rules to prove or refute a proposition e_p taking e_d for axioms. In word problems, u and $\bar{u}$ are the machines which apply the rewrite rules e_d to the initial word e_p.

(2) The universal machines u and $\bar{u}$ are defined in the system; the metaccode transformation μ is defined; and every machine and process which can be produced by substitution from the machines already defined is also defined. Atomic propositions about defined processes are legitimate propositions of the theory.

Goedel's incompleteness theorem. Under the assumptions (1) and (2) the formal system is incomplete if it is true.

(In the original Goedel's theorem the condition for the formal system was consistency, not truth).

Proof. Consider the machine G_1:

$$\langle G_1 e_d \rangle \rightarrow \langle u(e_d)*(G_1 \langle \mu e_d \rangle)? \rangle$$

It is defined in the theory because it is constructed as per-

mitted by the assumption (2). Substitute D for the argument in G_1: the resulting process $\langle G_1 D\rangle$ is again defined, therefore the proposition $\uparrow\langle G_1 D\rangle?$ is legitimate in the theory.

We shall say that the process A is *functionally equivalent* to the process B and denote this $A \equiv B$ if either both A and B are infinite, or both are finite and produce the same results (have the same final stage). From the definition of $\langle G_1 D\rangle$,

$$\langle G_1 D\rangle \equiv \langle u(D)*(G_1\langle \mu D\rangle)?\rangle$$

which can be rewritten as

$$\langle G_1 D\rangle \equiv \langle u(D)\uparrow\langle G_1 D\rangle?\rangle$$

From the deifinition of g, the right side can be replaced by a call of g:

$$\langle G_1 D\rangle \equiv \langle g\uparrow\langle G_1 D\rangle?\rangle \tag{3}$$

We see that $\langle G_1 D\rangle$ is the familiar Goedel process which proves in the theory its own infiniteness. Therefore, $\uparrow\langle G_1 D\rangle?$ is a true and legitimate proposition which, however, is not proved in the theory.▽

Goedel's theorem can also be proved using the other form of the Goedel process: a process which refutes its own finiteness. We define:

$$\langle G_2 e_d\rangle \rightarrow \langle \bar{u}(e_d)*(G_2\langle \mu e_d\rangle)!\rangle$$

Then $\langle G_2 D'\rangle$ is this process.

Practice. Give a full proof of Goedel's theorem using the G_2 machine.

The requirement of the interpretability of propositions in our theory makes it invulnerable to Goedel's argument and allows it to remain both complete and true. It is interesting that Goedel's theorem does not work on the two extremes of the power of a theory. On one extreme, a theory can be so weak that Goedel's process (or proposition) is impossible to express in it. In between we have all formal systems strong enough for Goedel's theorem. On the other extreme we have our theory, which forma-

lizes not one formal system, but an infinite sequence of formal systems. Since the functions γ and $\bar{\gamma}$ are changing in real time, we have to introduce the restriction of interpretability, which again makes the Goedel proposition unexpressible.

5. Metasystem transition

While Goedel's theorem is not applicable to our metatheory, it is, of course, applicable to every formal system we construct, if we replace the real-time human knowledge Γ by its definite stage Γ_i, i.e. a definite proposition, and make this change throughout the whole system. As a result of this operation, a *theory* of our metatheory, i.e. a real-time metamechanical process, will be converted into a purely mechanical device, a formal system in the usual sense, which can become an object of knowledge. We shall call this operation the *objectification* of the theory.

A simple way to objectify a theory is to cut off the access function gns from the world and define it by the sentence

(1) $\qquad \langle \underline{\text{gns}} \rangle \rightarrow \Gamma_i$

where Γ_i stands for the knowledge of the present stage of the theory. This, however, makes it impossible to use the objectified theory in the context of our general theory which goes on developing in real time, because we changed the definition of the function gns and, thereby, of all those functions which directly or indirectly call gns, including γ and $\bar{\gamma}$. Therefore, we have to make a *copy* of the theory and change in it the names of all, or at least some, functions to avoid confusion with the original functions of the general theory.

We come to the following procedure of objectification. Make a copy of the Refal machine implementing a theory. Redefine gns as in (1). Rename every Refal function in the copy, i.e. put in correspondence to every function symbol F a symbol F' never used before, and replace every entry of F when it immediately follows after an activation bracket $<$ by F'. In particular, rename γ as g and $\bar{\gamma}$ as $\bar{g}$; these new functions will be referred to as *the proving machines* of the objectified theory. Put all the new definitions into the program field of the Refal machine implementing the metatheory. This is, literally, a formalization of the an-

cient dictum: "Know thyself".

The formal system thus created is fully defined and represented by its two proving machines; now we can freely use them in our general theory. It is important to note that the renaming of functions in the process of objectification does not affect those function names which are used only as symbols in expressions; it is only when a function name *F* immediately follows an activation bracket that the effect of objectification becomes visible. Suppose a metacoded function call *(*FE*) is part of an expression processed by the functions of an objectified theory. It is processed in exactly the same way as it would be processed without objectification, until the time comes to demetacode the call through the use of functions step or act. At that time the modified versions step' and act' will be in action instead of the regular functions. They will operate as if *(*FE*) were demetacoded into <*F'E*>, not <*FE*>, without actually substituting *F'* for *F*.

To see it in more detail, remember the definition of the function step:

$$\langle \underline{\text{step}}\ *(\underline{\text{gns}})\rangle \rightarrow \langle \mu\ \langle \underline{\text{gns}}\rangle\rangle$$
$$\langle \underline{\text{step}}\ e_s\rangle \rightarrow \langle \underline{\text{stepu}}(P)e_s\rangle$$

The function stepu never calls gns, so we can leave it with the same name in the objectified theory. The modified function step' is defined by:

$$\langle \underline{\text{step}}'\ *(\underline{\text{gns}})\rangle \rightarrow \langle \mu\ \langle \underline{\text{gns}}'\rangle\rangle$$
$$\langle \underline{\text{step}}'\ e_s\rangle \rightarrow \langle \underline{\text{step}}(P)e_s\rangle$$

Here *P*, the current program, is an object expression, and is, therefore, the same in the modified definition as in the original. We see that if *F* is not gns the step is performed as if there was no modification. Nowhere is *F* replaced by *F'*. It is only gns which is modified because it is in the active position (follows an activation bracket) in the program.

The function act calls step repeatedly. The modified act' will call the modified step'. At each step of the emulated Refal machine, the function names in the metacoded function calls will remain unmodified; it is only when *(gns) is to be activated that the difference between the original theory and its objectified version manifests itself: instead of the current knowledge Γ, one

of the past stages of knowledge Γ_i is invoked. Since that time, human knowledge may have developed dramatically, but the objectified theory still operates as the general theory operated at that stage. And it "does not know" that it is objectified; in the metacode, the objectified theory uses the same language it used before, when it represented the present of the general theory. In particular, it refers to the cognitive functions as γ and $\bar{\gamma}$, although when they are activated they are executed with the fixed Γ_i, i.e. as g and $\bar{g}$. But it is only the metasystem, our general theory, that knows that things change in real time and has a different notation for the true cognitive functions on the one hand, and objectified cognitive functions, i.e. proving machines of various formal systems, on the other.

Looking through the list of the functions we defined we see that the following three functions refer directly to gns: step, γ and $\bar{\gamma}$. Those functions which call step, call gns indirectly, and so do the functions which call these functions, etc. The functions γ and $\bar{\gamma}$, renamed into g and $\bar{g}$, will be accessed from the general theory, but they are *not recursive* and *never called from inside* of the objectified theory.

This last detail may seem strange. We know that the process of proof in logic *is* recursive. So is the process of hierarchical generation of propositions in our theory. How is it possible that the proving machines of the objectified theory, i.e. a formal system, are not recursive?

The functions γ and $\bar{\gamma}$ in our theory may call themselves, but not directly (call as value) but through the metacode (call as process). The recursive configurations are not $\langle \gamma e_x \rangle$ and $\langle \bar{\gamma} e_x \rangle$, but $\langle$step $*(\gamma e_x)\rangle$ and $\langle$step $*(\bar{\gamma} e_x)\rangle$. When we rename cognitive functions into proving machines, their active entries are changed -- and they appear only once, at the time of definition -- whereas the metacoded entries are left unaffected. The new recursive configurations are $\langle$step' $*(\gamma e_x)\rangle$ and $\langle$step' $*(\bar{\gamma} e_x)\rangle$. Essentially, it is the function step' (since it depends on Γ_i, it should be denoted as step_i) which defines the operation of an objectified theory, and other functions are redefined in order to call step' instead of step. But when we are using the objectified theory as a formal system without going into the mechanics of it, we need only the proving machines g and $\bar{g}$.

Now that our general theory has the means to deal with its objectivized self, we can make a step in real time and expand our knowledge. Take the Goedel proposition for the objectified Γ_i in any of the forms discussed above; let us denote it G_i. Make the statement:

$$\Gamma \vdash G_i$$

It is true because, as we proved above, G_i is true. But it was not provable before we made this statement. Now, of course, it is trivially provable, because the current state of knowledge is

$$\Gamma_{i+1} = \underline{\text{and}}(\Gamma_i, G_i)$$

But it is a new formal system; these two propositions are true: $g_i(G_i)$? and $g_{i+1}(G_i)$!. The new system Γ_{i+1} can again be objectified and the proposition G_{i+1} added, etc.

We assume that the theories we are dealing with have a certain mimnimum of means (machinery and knowledge), which will be concretized as we go on. In particular, the conditions of Goedel's incompleteness theorem are met. We now have the following, somewhat schizophrenic, situation. There is the formal system F_i with the knowledge Γ_i. Among the machines defined in F_i there are g_i and $\bar{g}_i$, which mimic the operation of the cognitive functions of F_i. Also, there are the machines G_1 and G_2 defined, such that $G_1(\Gamma_i)$? and $G_2(\Gamma_i)$! are Goedel's propositions for the objectivized Γ_i (they will be denoted summarily as G_i):

$$\begin{aligned} G_1(\Gamma_i) &\equiv g(G_1(\Gamma_i)?) \\ G_2(\Gamma_i) &\equiv \bar{g}(G_2(\Gamma_i)!) \end{aligned}$$

We want to know what can and what cannot be proved in our theory at the present time, that is to say, in F_i. I call this situation schizophrenic because when we freeze the time and consider only the present moment, the difference between the formal system F_i and its objectified copy is only potential, not actual; the functions γ and $\bar{\gamma}$ on one hand, and g and $\bar{g}$ on the other, operate in exactly the same manner.

We proved that $g_i(G_i)$ cannot be finite. Therefore $\gamma(G_i)$ is not finite either: the Goedel propositions are not provable in F_i, though true. This much we knew before. Our aim now is to analyze in greater detail why the intuitive proof that G_i is true

cannot be formalized in F_i itself. This proof is short and simple. If there are no means at the present stage of Γ to formalize this proof, why cannot we add the necessary means to Γ once and forever?

Let us examine the proof of Goedel's theorem. First, we used the definitions of g_i and G_1 to prove (3). The definitional knowledge, as we discussed in Sec.5.2, can be always assumed to be present in the current formal system if it is mechanically universal. To be able to combine definitions as necessary for the proof, a formal system must use the transitivity of entailment and know how to make substitution. We suppose that the system Γ_i has this minimum of knowledge. Then:

(4) $\Gamma_i \rightarrow G_1(\Gamma_i)! \equiv g_i(G_1(\Gamma_i)?)!$

After this had been established, we reasoned as follows. From (4), if $G_1(\Gamma_i)!$ is true then $G_1(\Gamma_i)?$ is proven in the objectified version of Γ_i. Since it is a true formal system, $G_1(\Gamma_i)?$ must be true. The first step is simply the equivalence (4) read in one direction, therefore it is producible in Γ_i:

(5) $\Gamma_i \rightarrow$ <u>if</u> $G_1(\Gamma_i)!$<u>then</u> $g_i(G_1(\Gamma_i)?)!$

The second step uses the assumption that the objectivized version of Γ_i is true. It can be formalized by the proposition:

(6) <u>if</u> $g_i(P)!$<u>then</u> P

where P is an arbitrary proposition (metavariable). It states that if P is proved by g_i then P is true. This is, of course, a true proposition, but we do not know whether it is produced in Γ_i or not. Let us suppose that it *is* produced and see what happens if we go on with the proof. From (6) with $G_1(\Gamma_i)?$ substituted for P:

(7) $\Gamma_i \rightarrow$ <u>if</u> $g_i(G_1(\Gamma_i)?)!$<u>then</u> $G_1(\Gamma_i)?$

From (5) and (7) by transitivity of implication:

(8) $\Gamma_i \rightarrow$ <u>if</u> $G_1(\Gamma_i)!$<u>then</u> $G_1(\Gamma_i)?$

Now we can come to a contradiction in the same way that we came to a contradiction in Goedel's proof. If we add $G_1(\Gamma_i)$ to

Γ_i, then by (8), $G_1(\Gamma_i)$? will also be produced. This means that $G_1(\Gamma_i)$! is contradictory in Γ_i, i.e.

$$\Gamma_i \rightarrow \bar{\nu}(G_1(\Gamma_i)!)!$$

Since $G_1(\Gamma_i)$ is a mechanical process:

$$\Gamma_i \rightarrow G_1(\Gamma_i)?$$

This, however, contradicts the unprovability of $G_1(\Gamma_i)?$ in Γ_i, which we have already firmly established. We conclude that (6) is not produced in Γ_i. Using the proposition $G_2(\Gamma_i)!$ instead of $G_1(\Gamma_i)?$, we come to the analogous conclusion regarding $\bar{g}$: the proposition

(9) $\quad$ if $\bar{g}_i(P)!$ then $\bar{\nu}(P)!$

with an arbitrary proposition P, is true but unprovable in Γ_i. We shall refer to the conjunction of the propositions (6) and (9) as *the correctness statement* for Γ_i.

When we discover a new knowledge which is not currently produced by Γ, we add it to Γ. We can do this, of course, with (6) and (9), which would result in a new formal system Γ_{i+1}. But these statements depend on the specific proving machines g and $\bar{g}$, so to add to knowledge in this fashion we have to consider every formal system individually. Our wish was to introduce a general principle which would be applicable at any stage and add it once and forever. We see now that we cannot do it. If we try to generalize (6) and (9) by substituting the general cognitive functions ν and $\bar{\nu}$ for the proving machines g and $\bar{g}$, we come to the trivial propositions:

if $\nu(P)!$ then P
if $\bar{\nu}(P)!$ then $\bar{\nu}(P)!$

which add nothing to the knowledge.

It may seem strange that a proposition so obviously true as the conjunction of (6) and (9) should never be provable in the current system. This happens because we look at the current system F_i and its objectified copy $F_i^!$ from the metasystem, and we see that they are essentially identical. But for F_i its objectified copy $F_i^!$ is just a machinery, and in order to predict any-

thing about its workings, F_i must analyze the definition of $F_i^!$ as such, without comparing $F_i^!$ with itself, because there is no *itself* to compare with, other than the objectified copy $F_i^!$. This is not the question of machinery, of course, but the question of knowledge. We can compare Γ_i and Γ from the vantage point of our human metasystem, as creators of all theories. But Γ is a *meta-symbol*. It does not appear in the formalism; it is only the access function gns that is to be found there. And it is used in such a manner that the result of its concretization cannot be subject to analysis, but only used in the functions γ and $\bar{\gamma}$ (where it *works*, being hierarchically activated, so even there it is not treated as object).

We now turn to the second theorem proven by Goedel, which states that the consistency of a strong enough theory cannot be proven in the theory itself. How should this statement be formalized in our theory? First, let us try the following way. Take any proposition P; its provability is $\gamma(P)!$. Take its negation $\bar{\gamma}(P)!$; the provability of the negation is $\gamma(\bar{\gamma}(P)!)!$. It seems that the negation of the conjunction of these two provabilities:

$$\bar{\gamma}(\gamma(P)! \ \& \ \gamma(\bar{\gamma}(P)!)!)!$$

should be the adequate expression of the idea. But this proposition is easily provable (producible) in any Γ_i which includes the basic logic principles! It corresponds to the propositon

$$\sim(P \ \& \ \sim P)$$

in usual notation.

The fallacy of this interpretation is that γ in our theory refers not to a specific formal system, but to *any* formal system we are ready to trust. This corresponds to the concept of being *true* in the usual approach, not to the concept of being *provable*. To formulate Goedel's second theorem we again have to make distinction between a formal system and its objectivized copy.

Goedel's consistency theorem. If the formal system Γ_i is strong enough, the consistency of its objectified copy, i.e. the proposition

$$\bar{\gamma}(g_i(P)! \ \& \ \bar{g}_i(P)!)!$$

cannot be proven in Γ_i.

Proof. In addition to the conditions of the incompleteness theorem, we shall assume a certain minimum of knowledge in Γ_i. First of all, it is the verification principle. If Γ_i includes it, then

(10) $$\Gamma_i \equiv R \;\&\; \underline{\text{if}}\; A!\underline{\text{then}}\; A!$$

with some R and an arbitrary A. The second assumption is:

(11) $$\Gamma_i \rightarrow \underline{\text{if}}\; A!\underline{\text{then}}\; \underline{\text{imp}}([R \;\&\; \underline{\text{if}}\; A!\underline{\text{then}}\; Q] \rightarrow Q)!$$

with arbitrary A, R, and Q. If this assumption does not hold, we can make the corresponding statement, which will be true. The proposition in (11) follows directly from the definition of the function imp, which runs the first argument until it produces the second (if it does). The running of the if generator in the square brackets in (11) must produce Q if A is finite.

Substituting $A!$ for Q in (11) and using (10) we get:

$$\Gamma_i \rightarrow \underline{\text{if}}\; A!\underline{\text{then}}\; \underline{\text{imp}}([\Gamma_i] \rightarrow A!)!$$

The call of imp here is a call of g_i, by the definition of the latter. Abbreviate $G_2(\Gamma_i)$ to G_2, and substitute it for A:

(12) $$\Gamma_i \rightarrow \underline{\text{if}}\; G_2!\underline{\text{then}}\; g_i(G_2!)!$$

As we saw earlier,

$$\Gamma_i \rightarrow \underline{\text{if}}\; G_2!\underline{\text{then}}\; \bar{g}_i(G_2!)!$$

Combining it with (12) we have:

(13) $$\Gamma_i \rightarrow \underline{\text{if}}\; G_2!\underline{\text{then}}\; [g_i(G_2!)! \;\&\; \bar{g}_i(G_2!)!]$$

Suppose that the consistency of (the objectified copy of) Γ_i can be proven in Γ_i:

(14) $$\Gamma_i \rightarrow \bar{\forall}(g_i(P)! \;\&\; \bar{g}_i(P)!)!$$

Substituting $G_2!$ for P we get:

(15) $$\Gamma_i \rightarrow \bar{\forall}(g_i(G_2!)! \;\&\; \bar{g}_i(G_2!)!)!$$

If we add $G_2!$ to Γ_i, (13) and (15) will immediately produce opposite propositions. Therefore $\bar{\gamma}(G_2!)!$ will be provable in Γ_i. But this, as we know, leads to a contradiction. Therefore, (14) is false, which proves the theorem.▽

We found three propositions which are true but unprovable in Γ_i:
(1) Goedel's proposition in two equivalent forms: $G_1(\Gamma_i)?$ and $G_2(\Gamma_i)!$.
(2) The consistency statement for Γ_i.
(3) The correctness statement for Γ_i.
While the first two give us only individual propositions, the third is a general principle, which yields infinitely many individual propositions. One divines a great power in it. We saw earlier that when we add the correctness statement to Γ_i, the resulting system Γ_{i+1} proves Goedel's proposition. We show now that the consistency statement also follows from the correctness statement (which intuitively should have been expected).

From Γ_{i+1} by adding the correctness statement:

$$(\underline{\text{if}}\ g_i(P)!\underline{\text{then}}\ P)\ \&\ (\underline{\text{if}}\ \bar{g}_i(P)!\underline{\text{then}}\ \bar{\gamma}(P)!$$

Then

(16) $\Gamma_{i+1} \rightarrow \underline{\text{if}}\ g_i(P)!\underline{\text{then}}\ P$
(17) $\Gamma_{i+1} \rightarrow \underline{\text{if}}\ \bar{g}_i(P)!\underline{\text{then}}\ \bar{\gamma}(P)!$

Consider the proposition $g_i(P)!\ \&\ \bar{g}_i(P)!$. It follows immediately from (16) and (17) that

$$\Gamma_{i+1} \rightarrow \underline{\text{if}}\ \gamma(g_i(P)!\ \&\ \bar{g}_i(P)!)!\underline{\text{then}}\ (P\ \&\ \bar{\gamma}(P)!)$$

Hence by the contradiction principle:

$$\Gamma_{i+1} \rightarrow \bar{\gamma}(g_i(P)!\ \&\ \bar{g}_i(P)!)!$$

for any P.▽

Thus the correctness statement alone is sufficient to cover all known propositions of that kind. The objectification of the current knowledge and the statement of its correctness is a *metasystem transition* with respect to the current system; it allows us to expand our knowledge. The results of this section

concerning this method of expanding knowledge can be summed up in the following important theorem:

Metasystem transition theorem. We can expand the knowledge of the current formal system Γ_i by making an objectified copy of it and adding to Γ_i the correctness statement

$$(\underline{\text{if}}\ g_i(P)!\underline{\text{then}}\ P)\ \&\ (\underline{\text{if}}\ \bar{g}_i(P)!\underline{\text{then}}\ \bar{\gamma}(P)!$$

In the resulting formal system Γ_{i+1} the Goedel proposition for Γ_i and the consistency of Γ_i are proved.▽

6. Classical logic

We saw that the static interpretation of Γ leads to intuitionist logic. With the dynamic interpretation, we can add to human knowledge the law of the excluded middle in its general form:

(EM_2) $\qquad \underline{\text{all}}(p \in \underline{\text{prop}}: \underline{\text{or}}(\gamma(p),\bar{\gamma}(p))!)$

Then we get classical logic in full. The law of double negation is deduced immediately. Conversly, we could postulate the double negation law and deduce (EM_2).

The strength of classical logic as compared to intuitionist logic comes from the more permissive treatement of the cognitive functions. Intuitionistic logic fixes the human knowledge Γ into Γ_i, at least for the time of discourse. Classical logic bases its proofs on the concept of the growing Γ; it allows the index i in Γ_i to go into infinity. This invokes the type of reasoning familiar from the calculus: "for every x there exists such a y that..." etc. The relation between Goedel's theorem and the law of the excluded middle becomes very clear when seen in this light. Goedel's theorem establishes that:

(a) For every formal system Γ_i there exists such a proposition G_i that

(1) $\qquad \underline{\text{or}}(g_i(G_i),\bar{g}_i(G_i))?$

(b) For that very proposition G_i there exists another formal system Γ_{i+1}, such that

(2) $\underline{or}(g_{i+1}(G_i), \bar{g}_{i+1}(G_i))!$

If we simply take the limit of (1) and (2) for $i \to \infty$, 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 the generator of propositions in (EM_2) run infinitely. Then it will generate at least one proposition, namely G_i, such that the disjunction in (EM_2) is false, and hence (EM_2) is false. In classical logic we use γ and $\bar{\gamma}$ to denote the limit of g_i and $\bar{g}_i$ as $i \to \infty$. The generator (EM_2) produces

(3) $\underline{or}(\gamma(P), \bar{\gamma}(P))!$

for every proposition P. Thus P comes first, and now we interpret (3) by seeing Γ and $\bar{\gamma}$ as the corresponding limits. Then for every P there is an i for which (3) is true, and (EM_2) becomes true.

Negative results in mathematics exert a hypnotizing action on mathematicians. When it is proven that something that had been considered very desirable does not really exist, people convince themselves that they did not really want it, and go after something else. The most famous case of this kind is the discovery of inconmensurability by the ancient Greeks, which prevented them from developing the algebra of real quantities. The Greeks could not overcome the threshold of introducing a notation like $\sqrt{2}$; for them it would have been a contradiction in itself because it was proven that no number becomes 2 when squared.

After a while, however, it is discovered that we still can introduce a notation for "non-existing entities" and interpret it as ideal objects to which we can strive and which can be approximated by entities which undoubtedly exist. It was Descartes (see *Turchin 1977*) who did it for inconmensurable quantities: a discovery that transformed mathematics.

We can draw a parallel between Goedel's theorem and the inconmensurability theorem. Universal resolving procedures have been as desirable in this century, and as much in the spirit of the time, as numerical representation of geometric quantities was in the time of the Pythagoreans. Goedel's theorem and the subsequent results of the non-existence of the most important universal algorithms undermined the effort to ground mathematics on

the concept of the algorithm (ditto the resolving procedure, ditto the machine). Ever since, such trends of thought as intuitionism and constructivism have remained on the margins of mathematics. The work on the foundation of mathematics lost its *elan vital*.

Goedel's theorem became widely known, and its popular interpretation is purely negative. It is thought of as showing our inability to do something. But in fact the balance of the message of Goedel's theorem is strongly positive. In its negative part it shows that we cannot construct certain universal procedures because the idea of such a procedure is, after a closer examination, self-contradictory. This is not what we intuitively understand by inability. It should not discourage us any more than our "inability" to create a number which is greater than itslef. On the positive side, Goedel's theorem gives a constructive procedure to find a proposition which is unprovable in a given formal system, but is true and *is proven to be true*. It shows that using metasystem transition we can prove propositions which we could not prove before. This is an *ability*, not an inability. It is this positive aspect of Goedel's method that is capitalized on in our theory.

It is only the form, the appearance of Goedel's theorem that looks negative. One and the same result can be presented both as a negative and as a positive statement. You can say: "there exists no maximal whole number", which looks like a negative statement. But you can also say: "whatever is a whole number N, I can construct a greater number $N+1$", and this is a positive statement. It makes a better sense, because it goes deeper into the matter and better reflects the proof.

There is an analogy between our theory and the geometric algebra introduced by Descartes, and it is not superficial at all. Mechanical procedures in our theory correspond to rational numbers at the time of Descartes. They "really exist". The functions ν and $\bar{\nu}$, and the whole lot of functions that can be defined using them, correspond to *irrational* (note the word!) numbers; they "cannot be understood by reason", and "do not really exist". The law of the excluded middle, in both its forms, is a sort of equation for the non-existent ν, like $x^2=1$ is the equasion for the non-existent $\sqrt{2}$. We approximate ν by g_1, g_2 ..., etc., as $\sqrt{2}$ is approximated by 1, 1.4, 1.41 ..., etc. The act of metasystem transition corresponds to taking the next term in infinite series

of infinitesimal analysis. By this analogy, our method may be called *metasystem analysis*. The parallel notions and entities are summed up in the following table.

metasystem analysis	**algebra and calculus**
Mechanical processes	Rational numbers
Goedel's theorem	The Pythagoras theorem
Incomputability	Inconmensurability
(*) $\gamma(P)! \vee \bar{\gamma}(P)!$	(*) $x/2 = 1/x$
"There is no recursive function γ such that (*) is true for all P"	"There is no x such that (*) is true"
Set theory. Zermelo	Geometry. Euclid
"Set theory is richer than automata theory"	"Geometry is richer than arithmetic"
Symbolic notation for all decision processes	Descartes's *Geometry*: symbolic notation for all geometric quantities
The solution of (*) is: g_1, g_2, g_3, ..., etc.	The solution of (*) is: 1, 1.4, 1.41, ..., etc.
Metamechanical processes	Irrational numbers
All processes	All real numbers

CHAPTER 6

Set Theory

1. Extensionality and regularity

There are two possibilities with regard to the set of all objects of a theory: it may be defined by a mechanical generator at the outset, or it may be generated by a real-time process as the theory develops. Theories with a mechanical generator of all objects are known as first-order theories; they have been discussed in the preceding chapter. Set theory is a theory of the second kind. Its objects are processes themselves; more precisely, they are interpretable set generators. At an early stage of the development of set theory it becomes clear that there exists no mechanical process which could generate all objects of set theory. So, the set of all objects of set theory keeps expanding as we keep developing the theory. We shall denote the current stage of this real-time process by the metasymbol ∧, and the Refal function which provides access to it by <lgs> (from the Greek 'Logos'). Thus ∧ is a mechanical generator of all the objects of set theory introduced up to now.

The machines of a theory are normally defined in such a way that if their inputs are legitimate objects of the theory and interpretable propositions, then the resulting process is interpretable. This is certainly true for the machines we have defined up to now and shall define below. Since the primary machines of a theory must be defined explicitely, their number can be only finite. With a given mechanical generator of all legitimate objects, the set of all interpretable propositions of a theory can be generated by substituting enumerably many entities for the inputs of the primary machines (including, of course, the substitution of entities which themselves result from previous substitutions). If *G* is the generator of all objects of a theory, then the generator of all possible propositions will be referred to as prop(*G*). In a first-order theory, the set of all interpretable propositions is enumerable, i.e. generated by a mechanical process. (The set of all true propositions of a first-order

theory is still a real-time process). In set theory we can use the mechanical generator prop of interpretable propositions for every stage Λ_i of the real-time process Λ. Thus the set of all interpretable propositions in set theory becomes a real-time process prop(lgs), which develops together with lgs. There are two essentially independent real-time processes in set theory: lgs with its derivative prop(lgs), which yields the *language* of the theory, and gns, which yields its *knowledge*. The only connection between them is that at any stage gns must be a subset of prop(lgs).

Passing on to a formal exposition of set theory, we identify the concept of a set with the concept of a generator, and contend that any interpretation of an infinite set which cannot be reduced to a generating process is intuitively meaningless. 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.

When an object and a set are given, we must be able to establish whether the object is an element of the set. A straightforward solution to this problem would be given by the function elm, which we have already used before (Sec.3.3). The process

$$\langle \underline{\text{elm}}(\uparrow E)\ \underline{\text{of}}\ G \rangle$$

stops if and only if the expression E is among the expressions generated by G.

This staightforward 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 two different processes in Refal 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 the case of infinite sets, the equality of expressions as set representatives, unlike their physical identity, cannot be directly established. A reference to some proof, i.e. to a knowledge, once again becomes an implicit part of semantics. Let us denote by *E*=*E'* the proposition that the sets represented by the expressions *E* and *E'* are equal; we shall write out this proposition in a moment. Then

```
<? E=E'>
```

is the process of proving that set *E* is equal to set *E'*. Using this notation we define function el as follows:

```
<el(e.x) e *(e.g)>  =  <el(e.x) e <step *(e.g)>>
                          | <eqs(e.x)(e.y)>
<el(e.x) e (e.y)e.z>  =  s|
                          | <el(e.x) e e.z>
```

Here the auxiliary function eqs, the equality of set-theoretical objects, is defined as:

```
<eqs(e.x)(e.x)>  =  T
<eqs(*(e.x))(*(e.y))  =  <? *(e.x) = *(e.y)>
```

The process <el(*E*) e *G*>, (which is el(*E* e *G*) in free format notation) stops if and only if *E* is an element of *G*.

Equality of sets is the double inclusion:

(1) $(S=T) \equiv ([S \underline{\text{in}} T] \,\&\, [T \underline{\text{in}} S])$

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

(2) $(S \underline{\text{in}} T) \equiv \underline{\text{all}}(x \in S\colon \underline{\text{el}}(x \in T)!)$

We see recursion here: function el is defined using the relation of equality, which in its turn calls function el. Furthermore, this is a *semantic* recursion because it crosses the boundaries of cognitive function calls. Therefore, we must ensure somehow that using function el we get interpretable propositions only.

From the definition of function el we see that $\underline{\text{el}}(X \in S)$ may call $\gamma(X=Y)$ where Y is any element of S. Thus $\underline{\text{el}}(X \in S)$ is semantically dependent on all the propositions $X=Y$ with a Y from S. Using symbol >> to denote semantic dependence, we can represent this by the formula:

(3) $\underline{\text{el}}(X \in S) \;>>\; X=Y \;,\; Y \in S$

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

From (1) and (2) we derive two time sequences:

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

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

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

(5a) $\underline{\text{el}}(X \in S) \;>>\; \underline{\text{el}}(Z \in Y) \;,\; Y \in S,\; Z \in X$

(5b) $\underline{\text{el}}(X \in S) \;>>\; \underline{\text{el}}(Z \in X) \;,\; Y \in S,\; Z \in Y$

A process which is semantically dependent on itself (infinite semantic recursion) is uninterpretable. From (5a) we see that we are immediately in trouble when $X=Z$ and $S=Y$. Since Z is an element of S, the el process will be uninterpretable unless

(6a) never: $S \in S$

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 situations are prevented only if

(6b) never: $S \in Y \ \& \ Y \in S$

We call an *el-sequence* a sequence of sets

$$Y_1, Y_2, \ldots, Y_i, \ldots$$

such that for every $i>1$

$$Y_{i+1} \in Y_i$$

It is easy to see that (6a) and (6b) can be generalized into

(6c) never: there is such an el-sequence of sets
$Y_1, Y_2, \ldots, Y_n$
that $S = Y_1 = Y_n$

This condition is necessary for interpretability, but not sufficient. The sufficient condition for the interpretability of the el function is:

(7) an el-sequence of sets which starts with S can only be finite

Sets S satisfying (7) are known as *regular*. Condition (7) is *the criterion of regularity*.

Regularity Theorem. The process 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:

$(X,S) \ >> \ (X',S')$
$(X,S) \ >> \ (S'',X)$

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 last call.▽

Since the repetition of an argument pair in an el-sequence creates an infinite el-sequence, requirement (6c) is satisfied if set *S* is regular.

If a set is regular, all its elements are regular. Indeed, should an element *T* of set *S* not be 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. Conversely, if all elements of *S* are regular, *S* is also regular. This 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.

Now we limit the objects of our theory to ur-elements and regular sets only, which guarantees the interpretability of the el processes. 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) <self> = (*(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.

It follows immediately from the definition of regularity that there is no (regular) set generator that could produce all

regular sets. Indeed, if such a generator Λ existed, then Λ itself would be a regular set; but then it must produce itself, which is impossible. Therefore, the generator of all sets in set theory is not a mechanical but a metamechanical (real-time) process, which develops as we define new sets.

We access the generator of all legitimate objects of set theory, i.e. ur-elements and sets, through the function lgs. At every moment in real-time the process <lgs> yields a specific mechanical generator Λ_i, which produces all those sets that are *already known to be legitimate*, i.e. interpretable and regular. Note that in the free format notation, where we ignore metacode transformation, the symbol lgs can be understood both as standing for the process <lgs> which produces Λ, and as the process of generation represented by Λ. We shall understand 'the process lgs', or 'the set lgs' in the second sense (the first process is trivial). The metasymbol Λ has no place in the formalism of our theory, we only use it to speak about the theory. The symbol lgs, however, is part of the formalism, and we must make its interpretation and possible uses clear. First of all: does it represent a legitimate object of the theory? It is a set generator, and as a process it must be, by definition, interpretable. But is it a regular set?

A set is regular if at some stage of the development of theory it becomes producible by lgs. Symbol lgs stands for the current Λ. At no moment in real time is Λ producing Λ, and since only those sets produced at some time by Λ are regular, we come to the conclusion that lgs itself is not regular. This seems paradoxical, because we know that at any moment in time Λ is a regular set. The resolution of this paradox is in remembering that the generator represented by lgs changes each time that we establish the legitimacy of a new generator as a set, for we immediately include this new generator in lgs. This is also true with regard to Λ. We can use every stage Λ_i of Λ in our theory, but the moment we use it we include it in lgs, thus turning Λ into Λ_{i+1}. Before we made this step, Λ_i did not qualify as a set. After we did it, Λ_i qualifies, but it does not represent lgs any more. The new value of lgs is Λ_{i+1}, and it again does not qualify as a set.

Viewing this problem from the formal side we must ask ourselves: does some of the generators Λ produce this Refal expression: *(lgs) ? It depends on how we define <lgs>. The preceding

discussion suggests that we should define it so that this expression is never produced and lgs is not regular. In fact, we have no choice. Suppose we define every stage ∧ as producing *(lgs) in addition to all other set representations. Since *(lgs) is immediately replaced by ∧, this amounts to ∧ generating itself. Therefore ∧ becomes not regular. We can still try to use it, taking care to separate *(lgs) from all other products which constitute the universe of regular sets available at the time. But now at every stage of the development of set theory, lgs will produce a ∧ which is not regular, and therefore lgs is not regular again.

In terms of Von Neumann's axiomatization, lgs represents a class, not a *set* of objects. The difference between a set generator and a class generator is not in what they ultimately produce: both produce the same regular sets -- but in the role they play at the current (and every) stage of theory, i.e. in the current formal system. The class of all sets is an ever-nascent creature which produces all legitimate sets, but is not yet itself legitimized as a set.

2. 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 of language creation and knowledge: ∧ and r.

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.

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 produces the whole list in one step:

$$\langle \underline{fs}\ e_1 \rangle \rightarrow e_1$$

The argument e_1 is expected here to be a list, e.g., (A)(B). If e_1 is not a list, the result of fs will not have the structure required of a set generator; therefore, the fs so defined is not, generally, a set generator. This would not scare a pure mathematician who naivelly assumes that functions always get only such arguments for which they are meant. But it does not seem right to a computer scientist; we would prefer to add what is known as a syntax check, so as to be safe with regard to the format of the result. Therefore we redefine function fs as follows:

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

Now $\langle \underline{fs}\ e_1 \rangle$ is always a generator no matter what its argument e_1.

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 *(fs(A)(B)) in the strict notation of our theory.

The empty set is *(fs). The set

{A,{A,B}}

is in our theory

*(fs(A)(*V(fs(A)(B))))

∇

Having fs-constructor alone, we can demonstrate how set theory is continuously developed. To start, we must define the set of ur-elements. Let it consist of three symbols: A,B, and C. At this initial stage our language generator is

$$\langle \underline{lgs} \rangle \rightarrow (A)(B)(C)$$

Suppose we want to consider a set which has exactly two

elements: A and B, i.e. {A,B} in the usual notation. We can do this. We write

(1) *(fs(A)(B))

and simultaneously modify lgs. It is now

<lgs> → (A)(B)(C)(*(fs(A)(B)))

Suppose we want to consider the set

(2) {A,B,{B,C}}

We cannot do that immediately, because one of the intended elements, namely {B,C}, is not in lgs. So we must first consider

(3) {B,C} = *(fs(B)(C))

and prove that it is a legitimate set. If we succeed in this, as we do of course, we add (3) to the list defining lgs. Only after that do we legitimize (2) as a set and add it to the list. Now the definition of lgs is:

<lgs> → (A)(B)(C)(*(fs(A)(B)))(*(fs(B)(C)))
(*(fs(A)(B)(*V(fs(B)(C)))))

In these examples lgs was modified by adding one object at a time. We also can expand lgs including in one step an infinite number of objects. For instance, it is possible to write a generator *(G) which will produce all finite sets that can be formed using three ur-elements A,B,C, and the fs constructor. We can then include *(G) into lgs. To do this conveniently we must change the format of the lgs definition, using not a list of objects but a list of generators. The new format is:

<lgs> → <uni [defining list] >

where [defining list] is a list of generators, and uni (for 'union') is the machine which produces all those elements producible by any of the generators in the list. It is defined as follows:

$$\langle \underline{uni}(e_1)e_2\rangle \rightarrow g\,|\langle \underline{act}\ e_1\rangle\,|\langle \underline{uni}\ e_2\rangle$$

$$\langle \underline{uni}\rangle \rightarrow$$

Assuming that a definition of *(G) exists and that it is proven to be regular, we can redefine lqs as

$$\langle \underline{lqs}\rangle \rightarrow \langle \underline{uni}(*(\underline{fs}(A)(B)(C)))(*(G))\rangle$$

We can now use any finite set constructible as defined above without changing lqs.

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

1. See if *E* is generated by lqs. If it is, use it.
2. If *E* is not produced by lqs
   prove that *E* represents an interpretable generator
   and that whenever it produces a set *S*,
   every element *T* of *S* either was in lqs 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 lqs. You can use it now.

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 $xU\{x\}$. Thus the elements of this set are:

(4) $\emptyset$, $\langle\emptyset\rangle$, $\langle\emptyset,\langle\emptyset\rangle\rangle$, $\langle\emptyset,\langle\emptyset\rangle$, $\langle\emptyset,\langle\emptyset\rangle\rangle$, ... etc.

The union of two sets *A* and *B* in the strict notation is

$$*(\underline{uni}(\uparrow A)(\uparrow B))$$

The set $\{X\}$ consisting of the single element *X* is

$$*(\underline{fs}(\uparrow X))$$

To construct a generator producing (4), we want a machine inf(X) which in every step produces an *X* and calls itself with $XU\{X\}$ as

the next argument:

$$\underline{\text{inf}}(X) \rightarrow (X)\ \underline{\text{inf}}(*(\underline{\text{uni}}(\uparrow X)(\uparrow *(\underline{\text{fs}}(\uparrow X)))$$

To translate this semi-formal recursion equation into a Refal definition we replace metacoding symbols ↑ by actual metacode transformations:

$$\langle \underline{\text{inf}}\ e_x \rangle \rightarrow (e_x)\langle \underline{\text{inf}} *(\underline{\text{uni}}(\langle \mu e_x \rangle)(*V(\underline{\text{fs}}(\langle \mu \langle \mu e_x \rangle \rangle))))) \rangle$$

It is easy to see that with any interpretable argument e_x the process $\langle \underline{\text{inf}}\ e_x \rangle$ is interpretable, and if e_x is a regular set, then it produces only regular sets. Hence $\uparrow\langle \underline{\text{inf}}\ \emptyset \rangle$, i.e.

(5) $\quad *(\underline{\text{inf}}\ *V(\underline{\text{fs}}))$

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. In the free format notation it is:

(6) $\quad \underline{\text{set}}(x \in S:\ H(x)!)$

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 consist in proving a certain property P of x:

(7a) $\quad \underline{\text{set}}(x \in S:\ \gamma(P(x))!)$

or its negation:

(7b) $\quad \underline{\text{set}}(x \in S:\ \bar{\gamma}(P(x))!)$

The definition of the set function is:

$$\langle \underline{\text{set}}\ *Es_x \epsilon\ *(e_g)(e_p)! \rangle \rightarrow \langle \underline{\text{set}}\ *Es_x \epsilon \langle \underline{\text{step}}\ *(e_g) \rangle (e_p)! \rangle$$

$$\langle \underline{\text{set}}\ *Es_x \epsilon\ (e_1)e_2\ (e_p)! \rangle \rightarrow$$
$$g|\ \langle \underline{\text{if}} \langle \underline{\text{act}} \langle \underline{\text{sub}}(*Es_x \rightarrow e_1)e_p \rangle \rangle \underline{\text{then}}(\langle \bar{\mu} e_1 \rangle) \rangle$$
$$|\ \langle \underline{\text{set}}\ *Es_x \epsilon\ e_2(e_p)! \rangle$$

$\langle$set *Es$_x$ ϵ e$_z$(e$_p$)!$\rangle$ $\rightarrow$

The format of the set function in strict Refal is:

$\langle$set ϵ G (H)!$\rangle$

where V is a free e-variable in metacode, G is a generator, and H a search. It works as follows. The generator G is run step by step. Each time it produces an object x, this object is substituted into the search H and this 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

(8) T = set(x ϵ S: $H(x)$!)

is an interpretable and regular set if and only if the following three conditions are satisfied: (a) the generator S is interpretable, (b) the process $H(x)$ 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 the set S is regular (and interpretable -- this goes without saying) then the necessary and sufficient condition for (8) to represent a legitimate set is that the process $H(x)$ is interpretable for every possible element of S.

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

(9) T = set(x ϵ lgs: $H(x)$!)

Although lgs is not regular, it is interpretable, because at every stage in real time it is represented by an interpretable mechanical generator. So, condition (a) is satisfied. But (b) is not. Indeed, consider the case when the search $H(x)$ is $\nu(P(x))$ or $\bar{\nu}(P(x))$. 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

(10) R = set(x ϵ lgs: $\bar{\nu}(P(x))$!)

with

(1Ø') $P(x) = x \in x$

is of that type. For R to be interpretable, the property $P(x)$ must be interpretable for every $x \in$ lgs, and since $P(x)$ is within a $\bar{\nu}$-call this interpretability must be proven before and independently of the interpretability of R. That is, R is *semantically dependednt* on $P(x)$, and on x, for every $x \in$ lgs.

At first glance it may seem that we could prove the legitimacy of R by the following reasoning. lgs produces only regular sets; therefore $P(x)$ and $\bar{\nu}(P(x))$ are interpretable for every x. Then R is interpretable and regular. This reasoning, however, is faulty. It assumes unconditionally that lgs produces only legitimate objects. But this can be taken for granted only before we start considering formula (1Ø). The moment we define R, it becomes part of our language, which means that the generator lgs undergoes a change *in the process of proving*: it now produces R. We cannot assume that this new lgs produces only interpretable and regular sets: not before we prove it. But R is not interpretable because it is among the x's produced by lgs and therefore semantically depends on itself. The set construct cannot be used with the universal generator lgs. 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 declare 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. 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. The paradoxes as they are known are built on the propositions which we have shown to be meaningless.

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

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

where S is a definite regular set, then we can still 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 pow:

$$\text{pow}(S) \rightarrow \text{set}(x \in \text{lgs}: \nu(x \text{ in } S)!)$$

The process pow(S) is *weakly interpretable*. Its semantic map is presented in Fig 5.1. 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 pow(S) may or may not be elements of S, but S itself certainly is not an element of S, being regular. Thus

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

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

The pow constructor stands alone from all the other constructors we have defined. It calls the function lgs which provides access to the real-time process $\wedge$ representing our developing mathematical language. If S is infinite then there exists no mechanical 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 that "some infinities are more infinite than others" 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. No wonder that the only way to reconcile such a philosophical foundation with mathematical practice, in which

set theory turned out to be extremely useful, has been to accept pure and dogmatic formalism. The formalistic philosophy is, of course, no philosophy at all. It simply refuses to discuss what mathematical propositions mean; still worse, it discourages the fresh minds coming into mathematics from thinking about it.

We contend that the mathematical formalism of set theory can be completely interpreted in terms of intuitively clear and unambiguous concepts. We should simply look better into how we are using mathematics, how we create its objects and satisfy ourselves about its proofs. 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 pronouns '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 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 subject of mathematical knowledge is inseparable from the concepts that are used here.

3. The axioms of set theory

Using the set constructors we defined above and adding a few more we can prove all the axioms of the ZF system as theorems in the Refal metasystem. Note that there are no ur-elements in the ZF system.

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) $(Ex)(Ay)[\sim(y \in x)]$

This set is *(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) $(Eb)(Ax)[x \in b \equiv x \in a \ \& \ P(x)]$

This set is

$$b = \underline{set}(x \in a: P(x))$$

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

(PAIR) $(Ec)(Ax)[x \in c \equiv (x=a \vee x=b)]$

This set is

$$c = *(\underline{fs}(a)(b))$$

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) $(Eb)(Ax)[x \in b \equiv (Ey)[y \in a \ \& \ x \in y]]$

We have to introduce a new constructor to satisfy this axiom:

$$\begin{aligned} &\langle \underline{sum}\ *(e_x)\rangle \rightarrow \langle \underline{sum}\ \langle \underline{step}\ *(e_x)\rangle\rangle \\ &\qquad\qquad\qquad\qquad\quad \mid \langle \underline{act}\ *(\langle \Pi e_1\rangle)\rangle \\ &\langle \underline{sum}\ (*V(e_1))e_2\rangle \rightarrow g \mid \\ &\qquad\qquad\qquad\qquad\quad \mid \langle \underline{sum}\ e_2\rangle \\ &\langle \underline{sum}\rangle \rightarrow \end{aligned}$$

Now the desired set b is *(sum a).

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

(POW) $(Eb)(Ax)[x \in b \equiv x \ \underline{in}\ a]$

The set b is 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 *(inf*V(fs)).

The idea of a function, or operation, is expressed in logic by a formula $F(x,y)$ which has this property:

$$(\forall x)(\exists y)(\forall z)[F(x,z) \equiv y=z]$$

It states that for every x there is exactly one y such that $F(x,y)$ holds. As a set-theoretical object, the function is the subset of the Cartesian product $a \times b$ which includes exactly those pairs $\{x,y\}$ for which $F(x,y)$ holds. Set a is called the *domain* of the function, and set b its *codomain*.

In our theory the concept of function is even more fundamental than in set theory, because we can identify the function with the parametrized search. The parameter of the search is the argument x of the function, and the result of the search y is its value. Thus we identify the concept of function with the process which computes this function. The computation, of course, may include references to real-time processes Γ and Λ, in which case we have to replace them by their best approximations currently known in order to actually make computations. A function defined in this way is, generally, *partial*, i.e. for some arguments the search for the value may be infinite. A function is *computable* if there are no real-time calls in the defining process.

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

$$\langle \underline{\text{fun}}(e_x)X\rightarrow Y\colon\ e_f\rangle \rightarrow \langle \underline{\text{sch}}\ *Ey \in \underline{\text{lgs}}\colon\ \gamma(\langle \underline{\text{sub}}(\langle \mu e_x\rangle \rightarrow *Ex)e_f\rangle)\rangle$$

Here e_x is to be replaced by the argument of the function, and e_f by the proposition $F(x,y)$. The string X→Y indicates that the functional argument enters $F(x,y)$ as *EX, and the functional value as *EY. Function fun substitutes the (metacoded) value of

e_x for *EX in F(x,y) and searches among all legitimate objects up to date for such a replacement of *EY that F(*EX,*EY) can be proven.

We may also need a Refal generator which computes a function and outputs its value as its single element. For this end we define a trivial function 'generator from search':

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

The desired generator is now:

$$\langle \underline{gfs}\ \langle \underline{fun}(e_x)X\rightarrow Y:\ e_f \rangle \rangle$$

If a function is defined as a set-theoretical object, we can construct the corresponding search analogously. This search will need no references to the access function <lgs>, because it will be sufficient to look through the codomain of the function.

VIII. Axiom of replacement. The image of a set under an operation (functional dependence) is again a set. More precisely, if α is a set and F(x,y) is a formula such that for every x from α 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 $x \in \alpha$ exists such that F(x,y):

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

To provide for such a set we introduce a new set constructor ima ('image'), which runs through the set α, and for every x generates the corresponding y:

$$\begin{aligned}
&\langle \underline{ima}(*(e_\alpha))e_f \rangle \ \rightarrow \ \langle \underline{ima}(\langle \underline{step}*(e_\alpha))e_f \rangle \\
&\qquad\qquad\qquad\qquad\quad |\langle \underline{gfs}\langle \underline{fun}(\langle \Pi e_1 \rangle)X\rightarrow Y:\ e_f \rangle \rangle \\
&\langle \underline{ima}((e_1)e_\alpha)e_f \rangle \ \rightarrow \ g| \\
&\qquad\qquad\qquad\qquad\quad |\langle \underline{ima}(e_\alpha)e_f \rangle \\
&\langle \underline{ima}()e_f \rangle \ \rightarrow
\end{aligned}$$

The generator ima(α,F) is only weakly interpretable, because ima(α,F) will be immediately added to lgs and tried by function fun which has it inside a ν-call. However, the use of this generator in propositions, which occurs always through the mediation of function el (extensionality principle), will not lead to

uninterpretability. The proof that for every x from α there is exactly one y such that $F(x,y)$ is done before we construct ima(α,F), hence for no x is y identical with ima(α,F). When this new set is added to lgs and tried by fun, it will never lead to a successful result (i.e. a halt), because for every x there is no more than one corresponding y.Therefore ima(α,F) cannot produce itself. All other branches in this process are interpretable and the sets produced -- regular.

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

(REG) $\quad \alpha \neq \emptyset \rightarrow (Eb)[b \in \alpha \;\&\; (Ax)[x \in \alpha \rightarrow \sim(x \in b)]]$

If every element of α has another element of α 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 α. Since α is regular this is impossible.

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

Such a function is referred to as a *choice function*. We can try to construct a choice function as a machine which runs a generator (an element of α) and stops the moment the first element is produced. This element becomes the value of the function:

$$\langle \underline{\text{cho}}\ *(e_b)\rangle \rightarrow \langle \underline{\text{cho}}\ \langle \underline{\text{step}}\ *(e_b)\rangle\rangle$$
$$\langle \underline{\text{cho}}\ (e_1)e_b\rangle \rightarrow \langle \bar{\mu}\ e_1\rangle$$

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. 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 is corresponds to number 1. If b is uncountable it calls lgs, 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. So, function cho is not objectively interpretable. (A more detailed discussion of functional interpretability will be given in the next section).

The axiom of choice stands apart from the other axioms of set theory. While all other axioms can be proven by defining simple and natural conctructors, the axiom of choice requires some more sophisticated, and perhaps artiicial, construction. At the present time we leave the interpretation of the axiom of choice an open question. Probably, it can be achieved using the following idea. Modify the definition of lgs so that this generator does not only produce regular sets, but with every non-empty set produces also one of its elements. Then the choice function will pick up exactly this element as the representative of the set. By this trick we fix the real time when we pick up the first produced element of a set as the time when this set gets its legitimacy. Later, instead of running the function cho each time that we want a representative, we always pick up the one produced at the outset.

For one of the most important problems of set theory, the problem of its consistency, the interpretation of the axiom of choice is unimportant. As proven in *Goedel 1940*, if set theory without the axiom of choice is consistent, then adding the axiom of choice will still leave it consistent. (Goedel proved it for von Neumann's axiomatization, but the equivalence of this axiomatization to that by Zermelo-Fraenkel has been established). We have proved that every axiom of the ZF set theory, with the exception of the axiom of choice, is true in our interpretation. By Correctness theorem it follows that set theory without the axiom of choice is correct. By Corollary 3 of Correctness theorem, it is consistent. According to the above, it follows that the full set theory is consistent. This result remains valid even if we regard set theory as a purely axiomatic theory: the existence of a non-contradictory model is enough to prove its consistency.

The fact that we are able to prove the consistency of set theory does not contradict Goedel's proof that the consistency of a theory cannot be proved in itself, because the crucial aspect of our theory, the distinction between real-time and model-time processes, cannot be formalized in set theory.

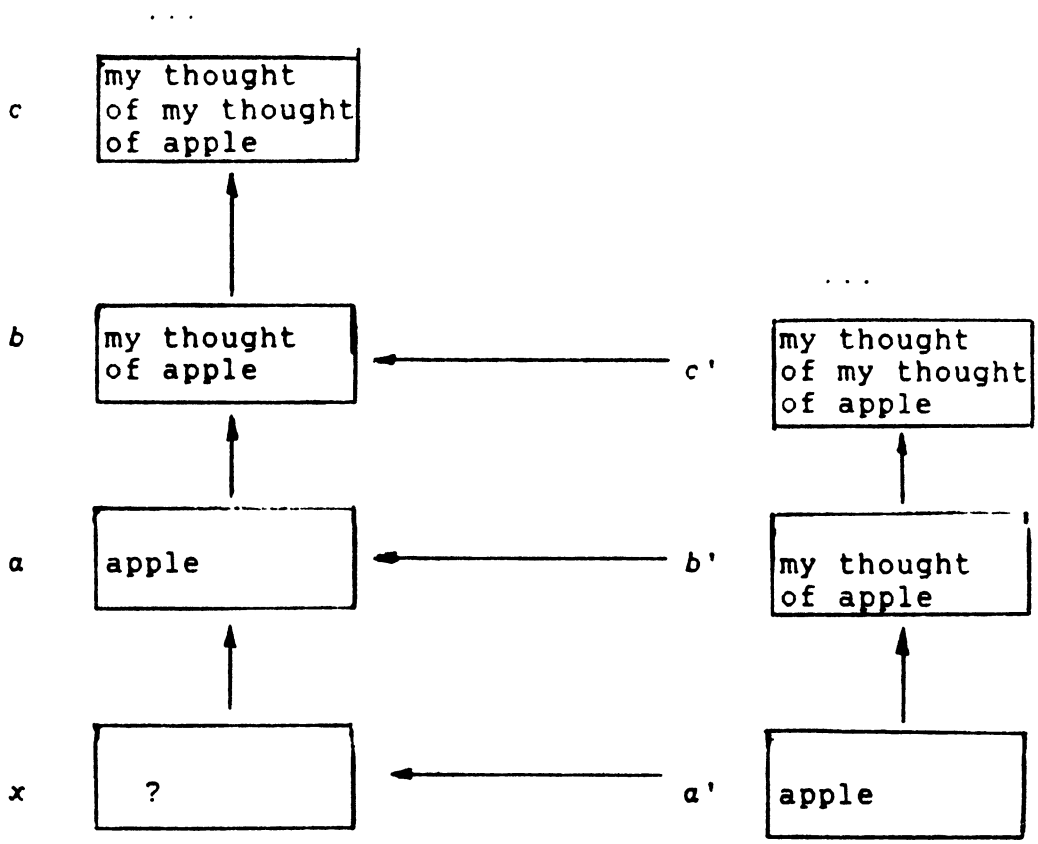

. . .
c
my thought of my thought of apple
b
my thought of apple
c'
. . .
my thought of my thought of apple
a
apple
b'
my thought of apple
x
?
a'
apple

Fig. 1.1

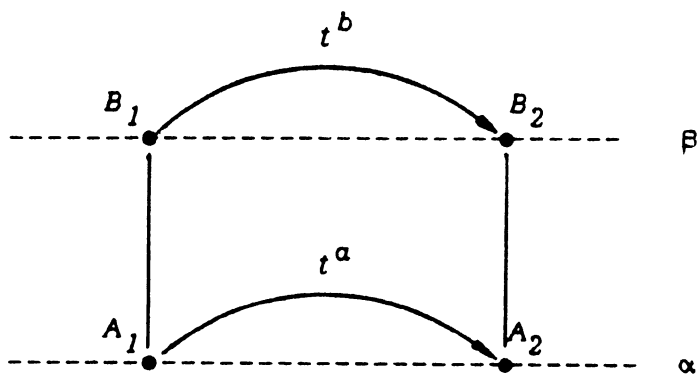

t^b
B_1
B_2
β
t^a
A_1
A_2
α

Fig. 1.2

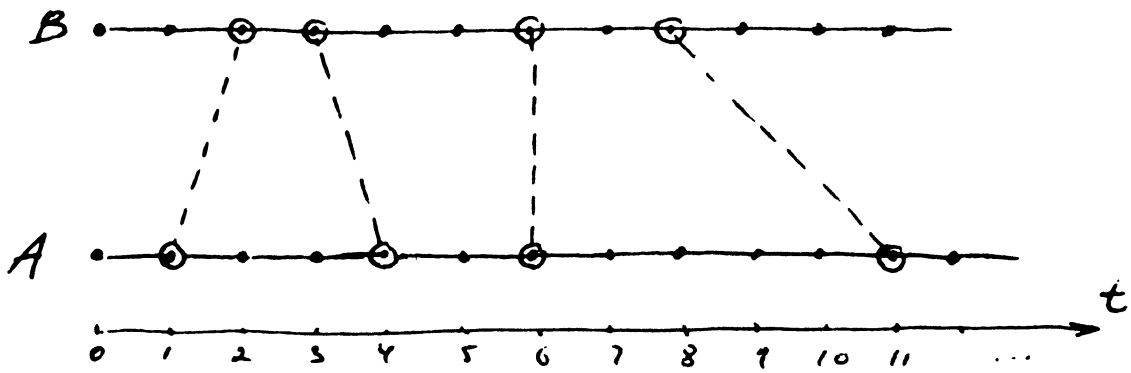


Fig. 3.1 Modeling relation between processes

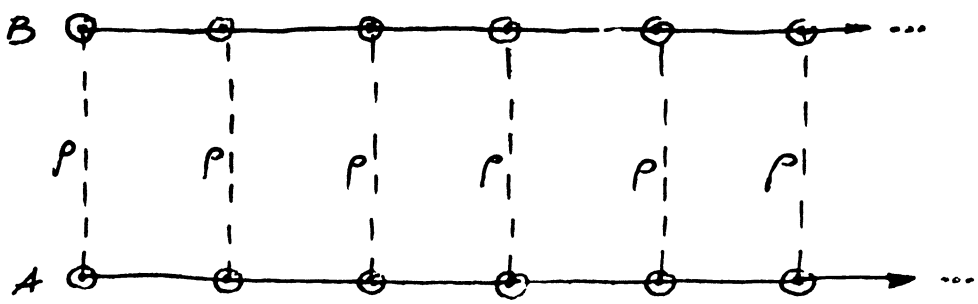


Fig. 3.2 The infinity model

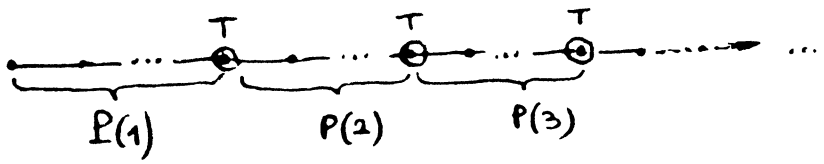


Fig. 3.3 . $(Ax)P(x)$

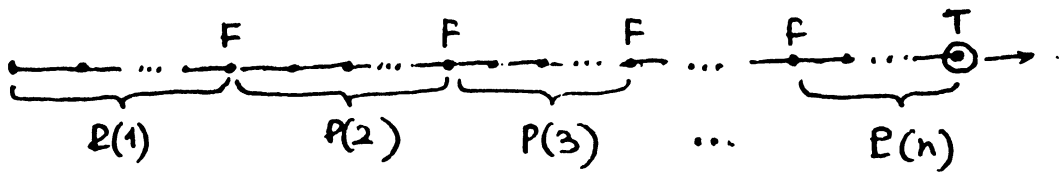


Fig. 3.4 $(Ex)P(x)$

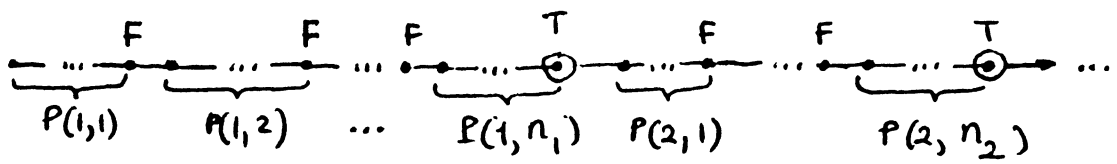


Fig 3.5 $(Ax)(Ey)\,P(x,y)$

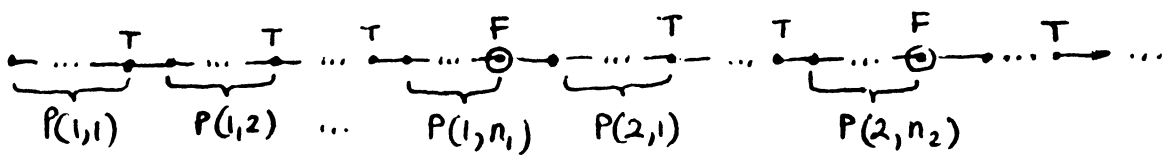


Fig. 3.6 . $(Ex)(Ay)\,P(x,y)$

To Chapter 4

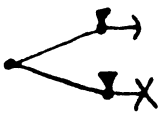


Fig. 1

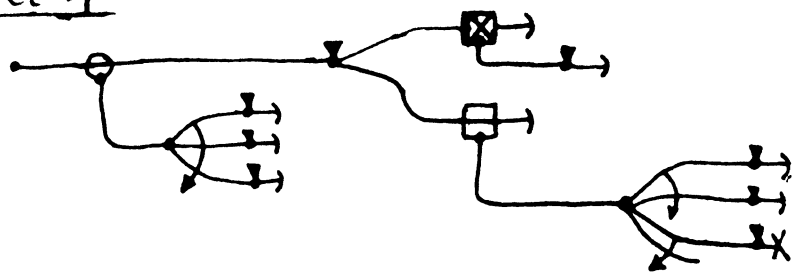


Fig. 2

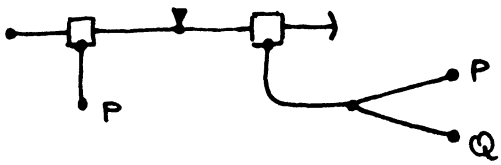


Fig. 3

Fig 4

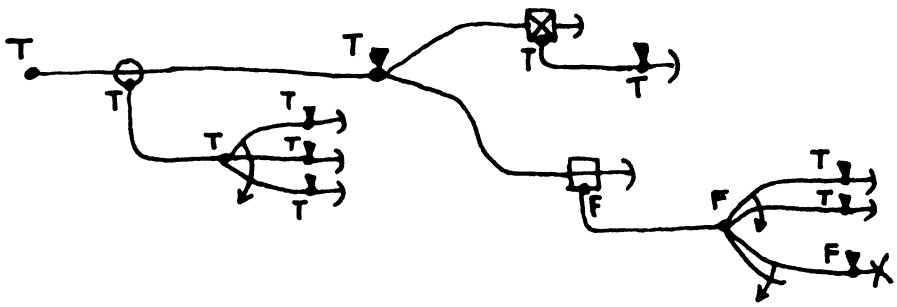


Fig. 5

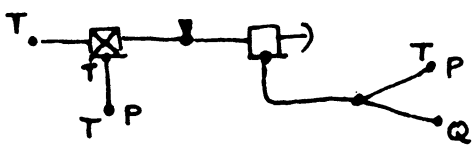


Fig 6a

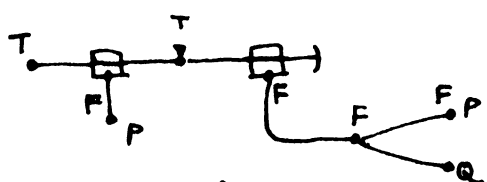


Fig 6b

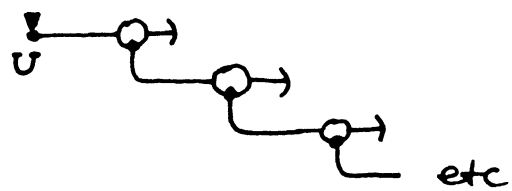
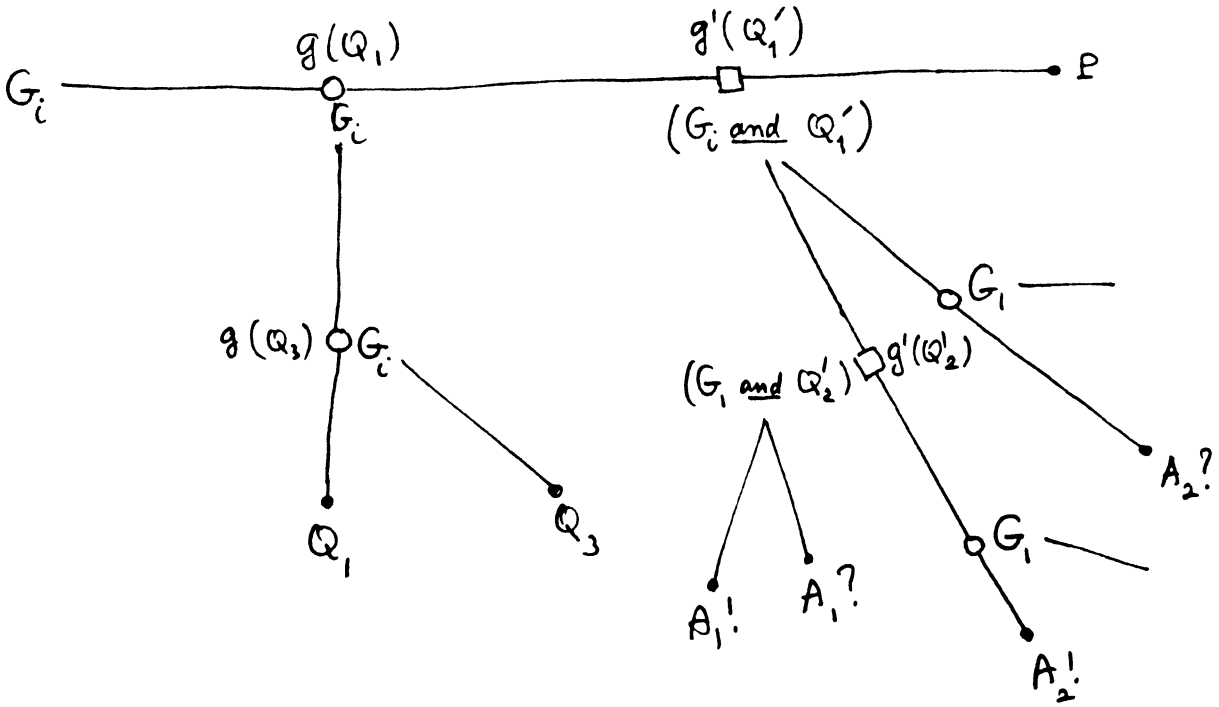


Fig 7



Chapter 4, Fig 8. Derivation tree

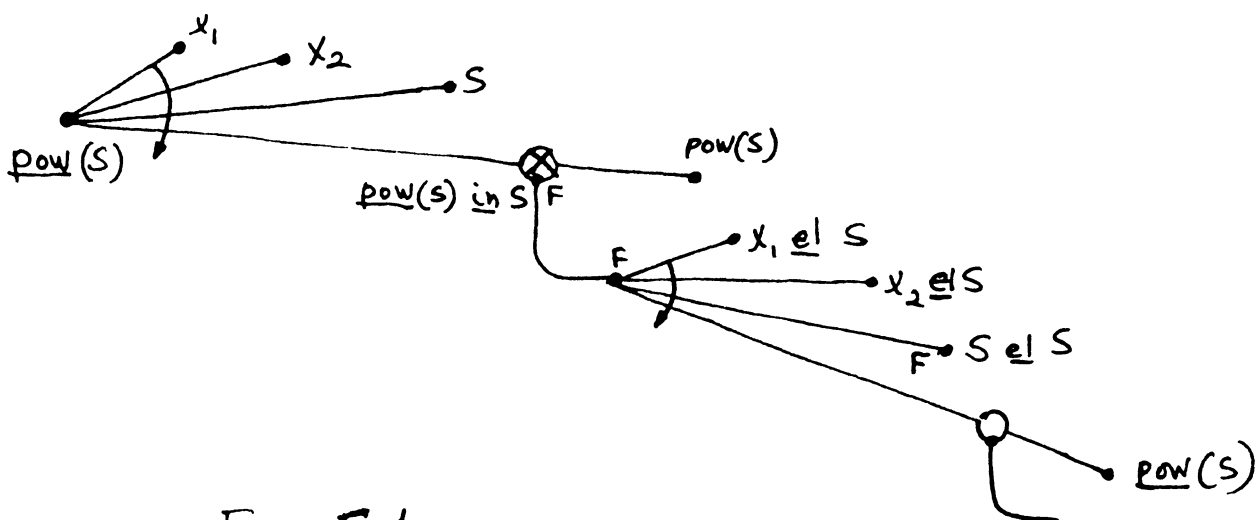


Fig. 5.1