TOWARDS THE THEORY OF PROGRAMMING IN CONSTRUCTIVE LOGIC

A.A.Voronkov
Institute of Mathematics
Universitetski Prospect 4
630090 Novosibirsk 90
USSR

Abstract. We develop an approach to the theory of extracting programs from proofs based on constructive semantics of the first order formulas called *constructive truth.* The underlying ideas are discussed. Using this notion of truth we define an appropriate notion of *constructive calculus.* Some results on relations between our theory and well known notions of constructive logic and the theory of enumerated models are proved.

1. Introduction. Extracting programs from proofs: the main ideas and problems

The extraction of programs from proofs or programming in constructive logic is based on the idea that under some restrictions proofs can be considered as programs. The general scheme of extracting programs from proofs is the following: at the beginning one writes a specification of the problem in some formal logical language (usually a variant of type theory). Then a formal proof of this specification is constructed and the program is extracted from this proof according to one of the known methods. Sometimes the last step can be absent because already the proof can serve as the program.

There are many distinctions between classical and constructive proofs. The main difference which allows one to consider constructive proofs as programs is the explicit definability property or $\exists$-property: if a proof Π of a closed formula $\exists xA$ is given then one can effectively construct a term t

such that $A(t)$ holds (in some appropriate sense). If there are free variables $x_1,\dots,x_n$ in A then such a proof can be considered as an algorithm meeting "specification" A (i.e. algorithm α such that $A(x_1,\dots,x_n, \alpha(x_1,\dots,x_n))$ holds for every $x_1,\dots,x_n$.

From the classical point of view the information sufficient for an adjustment of formula of the form $\forall\bar{x}\exists y\varphi(\bar{x},y)$ is the following: given any $\bar{x}$ one can find (generally speaking unclear in what way) an y such that $\varphi(\bar{x},y)$ hold. From the constructive viewpoint the adjustment of this formula means much more: there should be some general method (or *construction*) that allows to find such an y for given $\bar{x}$. The constructive proofs have the property that they implicitly contain an information sufficient for extracting such general method, which is in fact a program computing y by $\bar{x}$.

There are essentially two groups of methods of extracting programs from proofs. The first group uses syntactical methods like normalization of natural deduction proofs or cut elimination in sequential proofs. The second group is based on constructive semantics of formulas developed from Kleene's realizability. We briefly explain the basic ideas of these methods.

The normalization of proofs consist of syntactical transformations of the natural deduction proofs [Prawitz 1965] called *reductions*. The reductions are repeatedly applied to a proof while it is possible. The proof in the form where no application of reductions is possible is called normal proof or proof in normal form. Any normal proof of closed formula $\exists x\varphi(x)$ in e.g. intuitionistic predicate calculus or intuitionistic arithmetic takes the form

$$\frac{\begin{array}{c}\Pi\\ \varphi(t)\end{array}}{\exists x\varphi(x)}$$

Let a proof of a formula $\forall\bar{x}\exists y\varphi(x,y)$ is given:

$$\frac{\Pi}{\forall\bar{x}\exists y\varphi(x,y)}$$

Then one can use it as a program in the following way. Let $\bar{t}$ is a tuple of "input values" for the variables $\bar{x}$. To obtain an "output value" for y it is sufficient to normalize the proof obtained from the above proof by adding the $\forall$-elimination rule:

$$\frac{\begin{array}{c}\Pi\\ \forall\bar{x}\exists y\varphi(x,y)\end{array}}{\exists y\varphi(\bar{t},y)}$$

Its normal form takes the form

$$\frac{\begin{array}{c}\Pi_1\\ \varphi(\bar{t},s)\end{array}}{\exists y\varphi(\bar{t},y)}$$

Thus s is the intended value for y.

There are many lacks of normalization listed below.

(i) The proof is too large object to deal with and it is extremely inefficient to implement normalizations on proofs. So systems essentially based on normalization usually use not proofs but some structures coding only computationally useful information from proofs.

(ii) In the proofs of some particular theories there can be many introduction and elimination rules. In this case for each pair introduction rule - elimination rule for the same connective or quantifier it is needed to introduce new reduction rule and to prove normalization theorem for this extended calculus.

(iii) Reductions are not very expressible tool - as a matter of fact they are too simple. For example normalization lacks for the Markov's principle

$$\frac{\forall x(\varphi(x) \vee \neg\varphi(x)) \qquad \neg\neg\exists x\varphi(x)}{\exists x\phi(x)}$$

where x is variable ranging over natural numbers. The algorithm implicit in this rule is the following: using $\forall x(\varphi(x) \vee \neg\varphi(x))$ verify $\varphi(0)$, $\varphi(1)$, $\varphi(2)$...until a number n with $\varphi(n)$ is found. There is no reduction rule for the Markov's principle because to find such n one should normalize proofs of $\varphi(0) \vee \neg\varphi(0)$, $\varphi(1) \vee \neg\varphi(1)$...

In our opinion all this lacks of normalization lie in its syntactical nature. There is another techniques allowing to extract programs from proofs - realizability-like semantics of formulas. There are many such semantics developed from original Kleene's realizability [Kleene 1945]. These semantics reflect the constructive meaning of logical connectives first discovered by Kolmogorov [1932]. We consider the general scheme of realizability and discuss some lacks of existing realizability-like semantics.

Roughly speaking in this scheme we associate with any formula φ a relation $r \circledR \varphi$. If there is some element a with $a \circledR \varphi$ then we say that a "realizes" φ. The set of possible elements a depends on the used realizability. The idea is that such an a contains information which is sufficient to adjust φ. The general scheme of realizability is the following:

1. For atomic φ the relation $r \circledR \varphi$ is given and depends on the kind of realizability. Usually the "realizations" of atomic formulas are very simple elements (with no inner structure) and an atomic formula is realizable iff it is true or provable.

2. "Realizations" of $\varphi \wedge \psi$ are ordered pairs $<a,b>$ such that $a \circledR \varphi$ and $b \circledR \psi$.

3. "Realizations" of $\varphi \vee \psi$ are ordered pairs $<0,a>$ such that $a \circledR \varphi$ or ordered pairs $<1,b>$ with $b \circledR \psi$. (This is not particularly important that we choose 0 and 1 to distinguish between the two cases. Instead of 0 and 1 can be taken any two different elements of the set R of possible realizations.)

4. "Realizations" of $\varphi \supset \psi$ are "algorithms" α which given any a with $a \circledR \varphi$ give an output $\alpha(a)$ such that $\alpha(a) \circledR \psi$.

5. "Realizations" of $\exists x\varphi(x)$ are pairs $<t,a>$ such that $a \circledR \varphi(t)$.

6. "Realizations" of $\forall x\varphi(x)$ are "algorithms" α such that for every t $\alpha(t) \circledR \varphi(t)$.

This general scheme can be refined in many ways. For example in original Kleene's realizability for arithmetic instead of pairs and algorithms the Gödel numbers of these pairs and Kleene's number of the partial recursive function are taken. Realizability can be used for extracting programs from proofs as follows. Using the constructive proof of the formula $\forall\bar{x}\exists y\varphi(\bar{x},y)$ one can construct an element realizing this formula. By the above definition this element is an algorithm α such that for any tuple $\bar{a}$ of input values for $\bar{x}$ $\alpha(\bar{a})$ is a pair $<b,c>$ with $b \circledR \varphi(c)$. So c is the intended value for y.

Realizability seems more flexible than normalization in many aspects, but there are some lacks in existing definitions of realizability. First of all realizability-like semantics have one undesirable property: realizability (even in the case of arithmetic) contradicts to classical logic. There are classically false but realizable formulas. It means that the definition of calculus in which the proofs are constructed can not rely only upon realizability - if we do not want to obtain from the proof of $\exists x\varphi(x)$ an a

such that $\Phi(a)$ is false (but realizable).

The second undesirable property of realizability is that the usual definitions of realizability were designed for some special theories but there was not a definition suitable for large classes of theories. But if we are going to extract from proofs *programs* then the proofs must handle various data types: lists, numbers, functions etc. To treat such data types properly the notion of realizability is needed which can cover many various data types and many constructive theories describing properties of these data types.

Realizability and normalization have many common features. For example Mints [1974] proved that the two methods give equivalent programs in the case of intuitionistic predicate calculus. Similarly all we said about data types can as well be related to normalization.

One of the most important problems common for all existing approaches is the following: how to define constructive systems in which the proofs are constructed and how to construct algorithms for extracting programs from proofs? **To solve this problem one needs a general theory for programming in constructive logic - a theory which explains what is a constructive calculus and how such constructive calculi are related to programs to be extracted.** The existing approaches are either too particular or too general. Too particular means that it covers one particular calculus, such as arithmetic. To obtain generality all other theories are usually interpreted in this core theory. But to interpret e.g. lists in arithmetic is about the same as implementing list processing programs in machine codes. Too general means that one choose some very expressive type theory which allows to interpret everything and is (as any too general concepts) quite inefficient and unnatural.

In other words the following questions arise:

(1) what is the program? and

(2) what is the extraction of program? and

(3) what is the semantic relations between the proof and the extracted program?

The most general answer to the question (1) is that the program is an "algorithm". There is no problem with this answer because there are many well known approaches to the formal notion of algorithm and they are essentially (or more exactly extensionally) the same. The extraction of program is some algorithmic process transforming the proof to a description of algorithm. Proofs are conducted in formal systems. But the roots of formal systems are some domains of objects and proofs usually describe properties of these objects. For example proofs in formal arithmetic describe properties of natural numbers. If we agree with such treatment of proofs then we come to the following conclusion: the program extracted from a constructive proof of existence of an element with the desired property have to show a way to construct the element of the underlying model (or if we deal with a class of models then a way to construct the appropriate element for any model of this class). Since programs are algorithms then this way should be algorithmic. There is a generally adopted way to reason about the elements of these models - to encode or enumerate them.Thus we naturally come to the theory of enumerated models [Ershov 1980].

The notion of provability is secondary for models, the primary is the notion of truth. Thus the algorithm for extracting programs from proofs should be based on this notion. To implement such an approach the following things are to be done:

(1) It is necessary to express the truth of formulas algorithmically (the constructive encoding of formulas, or the constructive adjustment of truth).

(2) Proofs in formal systems should not give formulas that have no this constructive decoding;

(3) It is necessary to have an algorithm which constructs by the proof of a formula this constructive adjustment of truth of this formula.

In the rest of this paper we develop a formal theory intended to give the theoretical foundations for extracting programs from proofs based on above ideas. The more formal papers on this subject are [Voronkov 1988b, 1989b]. But a long list theorems as in [Voronkov 1989b] can try the most patient reader so here we will explain only the most essential results in this direction. The origins of such semantics can be traced to modified realizability [Kreisel 1959]. Very close to our classical realizability is the semantics studied by Läuchly [1970]. The first semantics suitable for several models was introduced in [Prank 1981]. Then Nepeivoda and Sviridenko [1982] proposed a semantics based on enumerable sets [Ershov 1977] but this paper contained some errors making definition of realizability incorrect. The correct presentation was given in [Voronkov 1985]. The similar semantics was independently discovered by Plisko [1987] but his semantics is closer to n-realizability from [Voronkov 1985] than to the semantics presented here.

2. The main auxiliary definitions

This section contains auxiliary definitions concerning the theory of enumerated models and higher type functionals. All definitions are given for one-sorted case but they can easily be generalized for many-sorted models.

Definition 1. Let S be a set. An *enumeration* of S is any mapping of the set of all natural numbers $\mathbb{N}$ onto S. An *enumerated set* is any pair (S,ν) where S is a set and ν is an enumeration of S.

Let $\mathfrak{M} = \langle M, P_0, P_1, \dots, f_0, f_1, \dots \rangle$ is a model of signature σ.

Definition 2. An *enumeration* of the model $\mathfrak{M}$ is any enumeration $\nu\colon \mathbb{N} \to M$ of the domain M of $\mathfrak{M}$ such that there exists a binary total recursive function F such that for any $n, y_1, \dots, y_{m_n} \in \mathbb{N}$

$$f_n(\nu y_1, \dots, \nu y_{m_n}) = \nu F(n, \langle y_1, \dots, y_{m_n} \rangle),$$

where $\langle y_1, \dots, y_{m_n} \rangle$ is the Gödel number of the tuple $y_1, \dots, y_{m_n}$. The pair $(\mathfrak{M},\nu)$ where $\mathfrak{M}$ is a model of signature σ, and ν is its enumeration is called an *enumerated model* of the signature σ.

Let $(\mathfrak{M},\nu)$ be an enumerated model of the signature σ and the (extended) signature Σ is the enrichment of σ with elements $\nu 0, \nu 1, \dots$ Using ν we can effectively construct some Gödel numbering μ of formulas of the signature Σ.

Definition 3. An enumerated model $(\mathfrak{M},\nu)$ is called *recursive* iff the set of all μ-numbers of quantifier-free formulas of the signature Σ is recursively enumerable. $(\mathfrak{M},\nu)$ is called *decidable* model iff the set of all μ-numbers of quantifier-free formulas of the signature Σ is decidable.

For correct definition of our semantics we need some formalization of functionals of higher types. As a suitable formalization we use elements of Scott's information systems [Scott 1982]. (Some another formalizations can be used as well, e.g. A-spaces [Ershov 1973] or f-spaces [Ershov 1977].) There is no place to write all formal definitions concerning information systems, so below we give only some informal explanations. Information systems allow one to define domains of functionals of higher types. For any information systems A,B there exist the information systems $A + B$, $A \times B$ and $A \to B$. We refer reader to [Scott 1982] for complete definitions. The set of all elements of an information system A, or simply the *domain* of A is denoted by A . In any information system A there always exists the least element $\perp_A$. The domain $A + B$ is essentially the disjoint union of the domains A and B plus the element $\perp_{A+B}$. The domain $A \times B$ consists of pairs $<a,b>$ with $a \in A$, $b \in B$. The elements of $A \rightarrow B$ are *continuous mappings* from A to B.

The elements of information systems are by definition sets of data objects. Suppose that we have a set of *elementary* information systems with natural numbers as data objects. Then set of data objects of the complex domains oonstructed from elementary ones according to definitions of $+$, $\times$, $\rightarrow$ is the subset of the set of *hereditarily finite sets over* $\mathbb{N}$. If we take some Gödel numbering of this set then we can speak about *computable elements.* An element (represented by some set A) is computable iff the set of Godel numbers of members of A is recursively enumerable.

The information system I_ω [Voronkov 1988b] is such that elements of I_ω are either $\perp$ or natural numbers $0, 1, \ldots$ The information system 1 is defined in such a way that it has the only bottom element $\perp_1 = \{0\}$.

3. Constructive truth

In this section we define a constructive semantics of first order formulas in such a way that the set of constructively true formulas is the subset of classically true ones.

Let $(\mathfrak{M},\nu)$ be an enumerated model of the signature σ. We assign to each formula φ of an extended signature Σ an information system A_φ and the relation $a \mathrel{c\ell} \varphi$ (a classically realizes φ) where $a \in A_\varphi|$. If φ is a formula with free variables $x_0,\ldots,x_n$ then $\forall\varphi$ will denote the formula $\forall x_0\ldots\forall x_n\varphi$. If there are no free variables in φ then $\forall\varphi \leftrightharpoons \varphi$.

Definition 4. (The relation $c\ell$). In (i)-(vi) all formulas are closed.

(i) $A_\varphi \leftrightharpoons 1$ for atomic φ and $\perp_1 \mathrel{c\ell} \varphi$ iff φ is true in $\mathfrak{M}$.

(Generally speaking the definition of $c\ell$ depend of the model $(\mathfrak{M},\nu)$, so we ought to write $c\ell_{(\mathfrak{M},\nu)}$ instead of $c\ell$, but we will omit indices where it will not cause ambiguities).

(ii) $A_{\varphi\wedge\psi} \leftrightharpoons A_\varphi \times A_\psi$ and $<a,b> \mathrel{c\ell} \varphi \wedge \psi$ iff $a \mathrel{c\ell} \varphi$ and $b \mathrel{c\ell} \psi$.

(iii) $A_{\varphi\vee\psi} \leftrightharpoons A_\varphi + A_\psi$ and, for $r \in A_{\varphi\vee\psi}$, $r \mathrel{c\ell} \varphi \vee \psi$ iff either for some $a \in A_\varphi$ $r = inl(a)$ and $a \mathrel{c\ell} \varphi$, or for some $b \in A_\psi$ $r = inr(b)$ and

$b\ \mathcal{cl}\ \psi$, where *inl* and *inr* are natural embeddings of A_φ and A_ψ into $A_{\varphi\vee\psi}$.

(iv) $A_{\varphi\to\psi} \leftrightharpoons A_\varphi \to A_\psi$ and, for $f \in A_\varphi \to A_\psi$, $f\ \mathcal{cl}\ \varphi \to \psi$ iff for every $a\ \mathcal{cl}\ \varphi$ $f(a)\ \mathcal{cl}\ \psi$.

(v) $A_{\exists x\varphi(x)} \leftrightharpoons I_{(\mathfrak{M},\nu)} \times A_{\varphi(t)}$, where t is an arbitrary closed term and $<n,a>\ \mathcal{cl}\ \exists x\varphi(x)$ iff $a\ \mathcal{cl}\ \varphi(\nu n)$.

(vi) $A_{\forall x\varphi(x)} = I_{(\mathfrak{M},\nu)} \to A_{\varphi(t)}$, and, for $f \in A_{\forall x\varphi(x)}$, $f\ \mathcal{cl}\ \forall x\varphi(x)$ iff for every $n \in \mathbb{N}$ $f(n)\ \mathcal{cl}\ \varphi(\nu n)$.

(vii) If there are free variables in φ then $A_\varphi \leftrightharpoons A_{\forall\varphi}$, and a cl φ iff $a\ \mathcal{cl}\ \forall\varphi$.

The following theorem explains why this semantics is called "classical realizability":

Theorem 5. There exist an a such that $a\ \mathcal{cl}_{(\mathfrak{M},\nu)}\varphi$ iff $\mathfrak{M} \models \varphi$. ■[1]

You may ask: so what is the need to introduce the new notion equivalent to the classical notion of truth? We made it because now we can easily define a constructive notion of truth such that the set of constructively true formulas are the subset of classically true ones:

Definition 6. Let $a \in A_\varphi$. Then $a\ \mathcal{con}\ \varphi$ iff $a\ \mathcal{cl}\ \varphi$ and a is computable. If $a\ \mathcal{con}\ \varphi$ then we will say that a *constructively realizes* φ. A formula φ is said to be *constructively true* in the enumerated model $(\mathfrak{M},\nu)$ (denoted by $(\mathfrak{M},\nu) \models_{con} \varphi$) iff there is an a with $a\ \mathcal{con}_{(\mathfrak{M},\nu)}\ \varphi$.

Thus we define the constructive notion of truth which does not diverse with the classical one, so the first part of our aim is satisfied. The second important property of our definition is that it is correct *for any denumerable model* and hence for any data type represented by such a model. Indeed, in [Voronkov 1986c] we investigated the properties of constructive truth for such an important data type as the type of lists with atoms from some model. It is also possible to generalize our definitions and technique for *parametric data types,* for example *lists(T),* where *T* is any data type, but in this case all needed definitions will take some space. (The papers on these developments are in preparation).

In the theory of enumerated models the most simple models are the decidable ones. The following theorem shows that if a model is "good" from the viewpoint of the theory of enumerated models then the constructive truth is identical to the classical truth for this model.

[1] We omit all proofs because of lack of space.

Theorem 7. Let $(\mathfrak{M},\nu)$ be decidable enumerated model, and φ be a formula. Then $(\mathfrak{M},\nu) \models \varphi$ iff $(\mathfrak{M},\nu) \models_{con} \varphi$.

The proof of this theorem is very simple but it shows that we are on the right way: for decidable models there is no difference between classical and constructive notion of truth. Some other results on relations between classical and constructive truth can be found in [Starchenko, Voronkov 1988] and [Voronkov 1988b, 1989b].

The next interesting question is the following: for what class of formulas the two notions of truth are equivalent independently of interpretation? This problem is closely related to the problem of eliminating computationally irrelevant parts from mechanically extracted proofs (see. e.g. [Goad 1980a,b, Henson 1989]). The following theorem partially gives an answer to this question:

Theorem 11. Let φ be a Harrop formula [Harrop 1960]. Then for every enumerated model $(\mathfrak{M},\nu)$ $\mathfrak{M} \models \varphi$ iff $(\mathfrak{M},\nu) \models_{con} \varphi$.

There are several another classes of formulas having this property but Harrop formulas are the most famous ones (see [Harrop 1960] and [Goad 1980b]).

4. Constructive theories

In this section we introduce the definition of constructive theory (constructive logic, constructive calculus) based on the above definition of constructive truth. We shown that the known first order theories are constructive in the precise sense of our definitions. It is proved that the intuitionistic predicate calculus is not complete for constructive truth. The very interesting result is given in Theorem 21: the intuitionistic arithmetic has only one (constructive) model (*categoricity* of intuitionistic arithmetic).

In what follows we will sometimes use informal notions (e.g. saying that the set of formulas is decidable). But all formal notions can of course be given.

Definition 9. The *inference figure* of signature σ is a finite sequence of formulas $\varphi_1,\dots,\varphi_n,\varphi$, where $n \geq 0$. The *calculus* $\mathfrak{L}$ is any effectively enumerable set of inference figures.

The *provability* of a formula in a calculus $\mathfrak{L}$ is defined as usual: φ is provable iff there is an inference figure of the form $\varphi_1,\dots,\varphi_n,\varphi$ in $\mathfrak{L}$ such that for all $i \in \{1,\dots,n\}$ φ_i are provable in $\mathfrak{L}$.

The following definition is given in informal terms:

Definition 10. The calculus $\mathfrak{L}$ is *constructive* for an enumerated model $(\mathfrak{M},\nu)$ iff there exists an algorithm α which by any inference figure $\varphi_1,\dots,\varphi_n,\varphi$ of $\mathfrak{L}$ and any $a_1,\dots,a_n$ such that $a_1 \mathcal{cl}\ \varphi_1,\dots,\ a_n \mathcal{cl}\ \varphi_n$ gives an a such that $a \mathcal{cl}\ \varphi$.

The following theorem shows soundness of the notion of constructive calculus w.r.t. constructive truth:

Theorem 11. Let $\mathfrak{L}$ be constructive for $(\mathfrak{M},\nu)$ and $\vdash_{\mathfrak{L}} \varphi$. Then $(\mathfrak{M},\nu) \models_{con} \varphi$. Moreover, an element a with $a\ \mathcal{con}\ \varphi$ can be found effectively by a proof of φ in $\mathfrak{L}$.

The application of Theorem 11 in program synthesis is immediate:

Theorem 12. Let $\mathfrak{L}$ be constructive for $(\mathfrak{M},\nu)$. Then given any proof Π of a closed formula $\forall\bar{x}\exists y\varphi(x,y)$ one can effectively construct a general recursive function g such that for any tuple $\bar{n}$ of natural numbers $\mathfrak{M} \models \varphi(\nu\bar{n},\nu g(\bar{n}))$.

To show that our definitions are generally applicable we have to show at least that the known constructive calculi are constructive in the sense of our definitions. The following theorems show the constructiveness of the intuitionistic first order predicate calculus, intuitionistic arithmetic (the analog of Nelson's theorem [Nelson 1947]) and constructiveness of the Markov's principle.

Theorem 13. Intuitionistic predicate calculus is constructive for every enumerated model $(\mathfrak{M},\nu)$.

Theorem 14. Intuitionistic arithmetic is constructive for the standard model of arithmetic $(\mathbf{N},\mathrm{id}_{\mathbf{N}})$.

Theorem 15. The calculus $\mathfrak{L}$ consisting of all instances of Markov's principle is constructive for every enumerated model $(\mathfrak{M},\nu)$[2].

Now we make some remarks about Markov's principle. In many papers on constructive logic and especially on program synthesis it is suggested that the constructive proof can have classical parts. (Many considerations but no formal definition on this subject are in [Goad 1980a,b].) There is one obs-

[2] Let us note: not only for natural numbers!

tacle to give semantics for such *mixed* proofs using traditional approaches because in the known realizability semantics there are formulas that are realizable but classically false. But in our semantics it is not so! Thus slightly changing our definitions we can prove the constructiveness of the following mixed inference figure[3]:

$$\frac{\vdash_{con} \varphi \vee \neg\varphi \qquad \vdash_{cl} \exists x\varphi}{\vdash_{con} \exists x\varphi}$$

The definition of constructive calculi depends of an enumerated model. It is interesting to describe the class of formulas which are constructively true in any enumerated model. This suggests the following definition.

Definition 16. The formula φ is called *constructively valid* iff for any enumerated model $(\mathfrak{M},\nu)$ $(\mathfrak{M},\nu) \models_{con} \varphi$.

Plisko [1988] showed that for a similar semantics the class of valid formulas is Π^1_1-complete and is hence even more complicated than the class of classically true formulas of arithmetic. However his proof can not be adapted to our semantics. All we are able to say now about the class of constructively valid formulas is the following theorem:

Theorem 17. There exist a constructively valid formula φ which is not provable in the intuitionistic predicate calculus.

There is one syntactical criterion of constructiveness which is (explicitly or implicitly) used in many papers:

Definition 18. A calculus $\mathfrak{L}$ is called *syntactically constructive* iff the following two conditions hold:

1. If $\vdash_{\mathfrak{L}} \varphi \vee \psi$, where φ,ψ are closed, then either $\vdash_{\mathfrak{L}} \varphi$ or $\vdash_{\mathfrak{L}} \psi$.

2. If $\vdash_{\mathfrak{L}} \exists x\varphi(x)$, where $\exists x\varphi(x)$ is closed, then for some term t we have $\vdash_{\mathfrak{L}} \varphi(t)$.

This definition is not very natural but one can hardly find the right definition based on syntactical considerations. We introduce a semantic criterion of constructivity:

Definition 19. A calculus $\mathfrak{L}$ is called semantically constructive iff there is an enumerated model $(\mathfrak{M},\nu)$ such that $\mathfrak{L}$ is constructive for $(\mathfrak{M},\nu)$.

There is one curious corollary of the last definition: the *classical*

[3] I do not who first proposed using this mixed principle. I learned it from N.N.Nepeivoda.

first order predicate calculus is semantically constructive! But why one can not use classical logical when studying e.g. finite models? This example shows that there are semantically constructive calculi which are not syntactically constructive. The following theorem shows that the converse is also true:

Theorem 20. There exists a calculus $\mathfrak{L}$ of a finite signature σ such that

1. there exists a model of $\mathfrak{L}$;
2. $\mathfrak{L}$ is syntactically constructive;
3. $\mathfrak{L}$ is not semantically constructive.

It is well known that the first order language is not very expressive from the model-theoretical point of view. For example any sufficiently rich theory have many non-isomorphic (countable) models. The following theorem shows that the constructive semantics can make the language more expressive:

Theorem 21. Any two constructive models of the Heyting arithmetic are recursively isomorphic.

Another results on expressibility of the language with constructive semantics are published in [Starchenko, Voronkov 1988, Voronkov 1988b].

References

Basin D.A. [1989]. *Building theories in Nuprl.* - In: Logic at Botic'89, LNCS 363, 1989, 12-25.
Bates J.L., Constable R.L. [1985] *Proofs as programs.* - ACM Trans. on Programming languages and systems, 1985, v.7, no.1, p.113-136.
Beeson M. J. [1978a] *Some relations between classical and constructive mathematics.* - J. Symb. Logic, 1978, v.43, no.2, p.228-246.
Beeson M.J. [1986] *Proving programs and programming proofs.* - In: VII Int. Congress on Logic, Methodology and Philosophy of Science, Elsevier Sci. Publishers, 1986, p.51-82.
Bertoni A., Mauri G., Miglioli P., Ornaghi M. [1984] *Abstract data types and their extension within a constructive logic.* - Semantics of Data Types, LNCS 173, 1984, 177-195.
Bishop E. [1970] *Mathematics as a numerical language.* In: Intuitionism and Proof Theory, North Holland, 1970, p.53-71.
Bresciani P., Miglioli P., Moscato U., Ornaghi M. [1986] *PAP - proofs as programs (Abstract).* - JSL 51(3), 1986, 852-853.
Chisholm P. [1987] *Derivation of a parsing algorithm in Martin-Lof's theory of types.* - Science of Computer Programming, 1987, v.8, no.1, p.1-42.
Constable R.L. [1972] *Constructive mathematics and automatic program writers.* - In: IFIP'71, North Holland, 1972, p.229-233.

Constable R.L. [1983] *Programs as proofs: a synopsis.* - Information Processing Letters, 1983, v.16, no.3, p.105-112.

Constable R.L. e.a. [1986] *Implementing mathematics with the Nuprl proof development system.* - Prentice Hall, 1986.

Coquand T., Huet G. [1988] *The calculus of constructions.* - Information and computation, 1988, v.74, no.213, p.95-120.

Dragalin A.G.[1967] *To the basing of Markov's principle.* - Soviet Mathematical Doklady, 1967, v.177.

Dragalin A.G. [1978] *Mathematical intuitionism. Introduction to the proof theory* (Russian). - Moscow, Nauka, 1978.

de Bruin N.G. [1980] *A survey of the project AUTOMATH.* - In: To H.B.Curry; Essays on Combinatory Logic, Lambda Calculus and Formalism, Academic Press, 1980, p.579-606.

Diller J. [1980] *Modified realization and the formulae-as-types notion.* In: Festschrift on the occasion of H.B.Curry's 80th Birthday, Academic Press, N.Y., 1980, p.491-502.

Ershov Yu.L. [1973] *The theory of A-spaces.* - Algebra i Logica, 12(4), 1973.

Ershov Yu.L. [1977] *The theory of enumerations* (in Russian). - Moscow, Nauka, 1977.

Ershov Yu.L. [1980] *Decidability problems and constructive models* (in Russian). - Moscow, Nauka, 1980.

Feferman S. [1979] *Constructive theories of functions and classes.* - In: Logic Colloquium'78, North Holland, 1979, p.159-224.

Feferman S. [1984] *Between constructive and classical mathematics.* - In: Computation and proof theory, Lecture Notes in Math 1104, 1984, p.143-162.

Goad C.A. [1980a] *Proofs as descriptions of computation.* - 5th CADE, LNCS 87, 1980, p.39-52.

Goad C.A. [1980b] *Computational uses of the manipulation of formal proofs.* - Stanford Univ. Department of CS, TR no. STAN-CS-80-819, 1980, 122p.

Gödel K. [1958] *Uber eine noch nicht benutzte Erweiterung des finiten Standpunktes.* - Dialectica, 1958, v.12, no. 3/4, p.280-287.

Goto S. [1979b] *Program synthesis from natural deduction proofs.* - 6th IJCAI, 1979, v.1, p.339-341.

Harrop R. [1960] *Concerning formulas of the types* $A \rightarrow B \vee C$, $A \rightarrow (Ex)B(x)$ *in intuitionistic formal system.* - J. Symb. Logic, v.17, 1960, pp.27-32.

Hayashi S., Nakano H. [1988] *PX: a computational logic.* - MIT Press, 1988.

Henson M.C. [1989a] *Information loss in the programming logic TK.* - Draft, Univ. of Essex, Department of Computer Sci., October, 1989.

Henson M.C. [1989b] *Realizability models for program construction.* - In: J.L.A. van de Snepsheut (Ed.), Mathematics of Program Construction, LNCS 375, 1989, 256-272.

Huet D. [1986] *Computation and deduction.* - Carnegie Mellon Univ., 1986.

Kleene S.C. [1952] *Introduction to metamathematics.* - Van Nostrand P.C., Amsterdam, 1952.

Kleene S.C. [1973] Realizability: a retrospective survey. - LNM 337, 1973.

Kolmogorov A.N. [1932] *Zur Deutung der Intuitionistischen Logik.* - Mathematische Zeitschrift, v.35, 1932, p.58-65.

Kreisel G. [1958] *Interpretation of analysis by means of constructive functionals of finite types.* - In: Constructivity in Mathematics, North Holland, 1958, p.101-128.

Läuchly H. [1970] *An abstract notion of realizability for which intuitionistic predicate calculus is complete.* - In: Intuitionism and Proof Theory, North Holland, 1970, p.227-234.

Manna Z., Waldinger R. [1979] *Synthesis: dreams -> programs.* - IEEE Trans. Software Engineering, 1979, v.5.,no.4, p. 294-328.

Markov A.A., Nagorny N.M. [1986] *Theory of algorithms.* - Moscow, Nauka, 1986.
Miglioli P., Moscato U., Ornaghi M. [1989] *Semi-constructive formal systems and axiomatization of abstract data types.* - TAPSOFT'89, LNCS 351, 337-351.
Mints G.E. [1974] *E-theorems* (in Russian). - Zapiski Nauchnyh seminarov LOMI, 1974, v.40, p.110-118.
Nelson D. [1947] *Recursive functionals and intuitionistic number theory.* - Trans. Amer. Math. Soc., v.61, 1947, p.307-368.
Nepeivoda N.N. [1979a] *An application of proof theory to the problem of con structing correct programs.* - Kibernetika, 1979, no.2.
Nepeivoda N.N. [1979b] *A method of constructing correct programs from correct subprograms.* - Programmirovanie, 1979, no.1.
Nepeivoda N.N., Sviridenko D.I. [1982] *Towards the theory of program synthesis (in Russian).* - In: Trudy Instituta Matematiki, v.2, Novosibirsk, Nauka, 1982, p.159-175.
Nordstrom B., Peterson K. [1983] *Types and specifications.* - IFIP'83, NH, 1983, 915-920.
Nordstrom B., Smith J.M. [1984] *Propositions, types and specifications of programs in Martin-Lof's type theory.* - BIT, 24(3), 1984, 288-301.
Plisko V.E. [1987] *On languages with constructive logical connectives.* - Soviet Mathematical Doklady, v.296, no.1., 1987.
Prank R.K. [1981] *Expressibility in elementary theory or recursively enumerable sets with realizability logic.* - Algebra i Logica, 20(4), 1981.
Prawitz D. *Natural deduction.* - Stockholm, Almquist and Wicksell, 1965.
Petersson K. [1982] *A programming system for type theory.* - LPM Memo 21, Dpt of CS, Chalmers Univ. of Technology, Goteborg, 1982.
Scott D.S. [1982] *Domains for denotational semantics.* In: Lecture Notes in Computer Science 140, 1982, pp.577-612.
Starchenko S.S., Voronkov A.A. [1988] *On connections between classical and constructive semantics.* - In: COLOG-88 (papers presented at the Int. Conf. on Computer Logic), Part 1, Tallinn, 1988, p.101-112.
Voronkov A.A. [1985] *A generalization of the notion of realizability.* - Unpublished Manuscript, 25pp. (Abstract was published in Siberian Mathematical Journal, 1986).
Voronkov A.A. [1986a] *Logic programs and their synthesis* (Russian). Preprint no. 23, Institute of Mathematics, Novosibirsk, 1986.
Voronkov A.A. [1986b] *Synthesis of logic programs* (Russian). Preprint no. 24, Institute of Mathematics, Novosibirsk, 1986.
Voronkov A.A. [1986c] *Intuitionistic list theory* (Russian). - In: Abstracts of the 8th All-union Conf. on Mathematical Logic, Moscow, 1986.
Voronkov A.A. [1987a] *Constructive logic: a semantic approach.* - In: Abstracts of the 8th Int. Congress on Logic, Methodology and Philosophy of Sci., v.5, part 1, Moscow, Nauka, 1987, p.79-82.
Voronkov A.A. [1987b] *Deductive program synthesis and Markov's principle.* - In: Fundamentals of computation theory, LNCS v.278, 1987, p.479-482.
Voronkov A.A. [1988a] *Constructive calculi and constructive models.* - Soviet Mathematical Doklady, 299(1), 1988.
Voronkov A.A. [1988b] *Model theory based on the constructive notion of truth.* - In: Model Theory and Applications, Trudy Instituta Matematiki, v.8, Novosibirsk, 1988.
Voronkov A.A. [1989a] *Program synthesis using proofs in first order logic.* (Russian). - To be published in Vychislitelnye sistemy, 1989, 50pp.
Voronkov A.A. [1989b] *A theory for programming in constructive logic.* - presented to the 3rd Int. Conf. on Computer Sci. Logic, Kaiserslautern, October, 1989.