Aleksei L'vovich Semenov
A simple detailed proof for Goedel's incompleteness theorem

Kybernetika, Vol. 24 (1988), No. 6, 447--451

Persistent URL: http://dml.cz/dmlcz/124187

KYBERNETIKA — VOLUME 24 (1988), NUMBER 6

A SIMPLE DETAILED PROOF FOR GOEDEL'S INCOMPLETENESS THEOREM

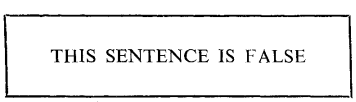
ALEXEI L. SEMENOV

The paper brings a simple detailed proof of Goedel's Incompleteness Theorem exposed in the language of informatics.

Goedel's Incompleteness Theorem is one of the central theorems of mathematical logic and many attempts at explanation have been done to make it accessible to a wide circle of mathematiciens and to the uses in applications, in computer science, e.g., see [1], [4] and [5]. These attempts are partially motivated by the importance of this result, negative in nature, for practical programming. It helps the programmer like other negative results from logic and algorithm theory, to make reasonable plans and not to desire the impossible. (See [2].) On the other hand, informatics give a proper context, formulation and explanation to the theorem. The present version of the proof is the clearest from the point of view of contemporary computer science. I used it successfully in my lectures at Moscow University, namely for the students of structural and applied linguistics.

As it is well known, the Liar's Paradox (see Fig. 1) can be considered as an origin of Goedel's Incompleteness Theorem.

Liar's Paradox



Is this sentence true?

Fig. 1.

The root of the paradox is in the presumption that each sentence must be true or false, which is not the case for the considered sentence. But for formal systems

studied by mathematical logic, each sentence is true or false. So if we imagine that there exists a sentence which expresses its own invalidity, we will get the same contradiction. So this sentence is impossible. (This is a sketch of Tarski's Theorem.) But let us substitute *unprovable* for *false*. Then we get Goedel's sentence (Fig. 2).

Goedel's sentence

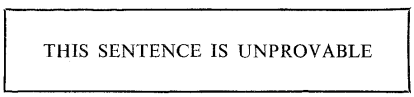


Fig. 2.

If this sentence is true, it is unprovable. If it is false, it is provable. In both cases, the notion of validity does not coincide with the notion of provability. But this is a formulation of Goedel's Incompleteness Theorem.

Goedel's Incompleteness Theorem. The notion of 'truthness' does not coincide with the notion of provability.

To prove Goedel's Theorem, we have to choose a language and a notion of provability. (Note that we have a standard notion of truth for our languages.) Goedel chose the language of elementary arithmetic and some notion of provability based on Peano's axioms.

Next, we must find a way to express the notion of provability, by means of our language. Then we must write down Goedel's sentence. Goedel's choice was very natural in the 30's, when the natural numbers were the main mathematical object. In fact, Goedel had to invent arithmetization – the method of coding formulae (which are in fact words) and sequences of formulae by natural numbers. But it was clear to Goedel that the language can be chosen in different ways. Today, words are commonly used in mathematics and in computer science, all information including quantitative is coded by zeros and ones. So we can choose some set of words (with natural operations on them) as the basic set of our theory, instead of natural numbers. On the other hand, there is no need to formulate any particular notion of provability – we can prove Goedel's Theorem simultaneously for all possible notions of provability.

In our proof, we shall also use coding for words over an arbitrary alphabet by words over a standard alphabet. For example, for digits we can choose the following codings:

0	010
1	0110
2	01110
3	011110
.....	
9	011111111110

The code of a word is the concatenation of its letters' codes:

$$020 \qquad 01001110010$$

Assuredly we can code some sequence of 0's and 1's as well as another sequence of digits.

Now we shall introduce some artificial operation, S, on words over an (ordered) alphabet with our coding. Operation S does not change any word containing some letter different from 0 or 1. To perform the operation on word, say, w over $\{0, 1\}$, we must:

1. code w, get some word $\tilde{w}$;
2. find a first occurrence of 010 (code of the first letter of the alphabet) in w:
   $w = \ldots 010u$
   substitute $\tilde{w}$ for 010:
   $\ldots \tilde{w}u$
   then find a first occurrence of 010 in u and substitute $\tilde{w}$, etc.

The word obtained by this sequential substitution is the result of S applied to w.

We choose the following alphabet for our language:

$$x\,0\,1\,\Delta\;S = (\;)\;\neg\;\vee\;\&\;\rightarrow\forall\;\exists y z p q s t u v$$

The list is finite, but it is irrelevant for our needs how to fix it; x, y, z are variables, S – unary functional symbol. x is in the first place for some technical reason. The terms of the theory are constructed in the following way:

$$\begin{array}{ll} 0,\ 1,\ \Delta & -\ \text{terms} \\ x,\ y,\ z,\ \ldots & -\ \text{terms} \\ \hline \alpha,\ \beta & -\ \text{terms} \\ \hline \alpha\beta,\ S(\alpha) & -\ \text{terms} \end{array}$$

Atomic formulae have the form $\alpha = \beta$, where α and β are terms. Arbitrary formulae are constructed in the ordinary way.

The interpretation of our language is simple. The domain is the set of all words over $\{0, 1, \Delta\}$, $\alpha\beta$ is the ordinary concatenation of words, $S(\alpha)$ is the operation described above.

Now we proceed to the notion of provability. First of all we introduce the standard notion of grammar. A *grammar* is the tuple $G = \langle T, N, A, \Pi\rangle$, where T is the (finite) terminal alphabet, N – the (finite) nonterminal alphabet, $A \in N$, A is the initial symbol, Π – the finite set of productions of the form $\varphi \rightarrow \psi$, where φ, ψ are words over $T \cup N$. The word w over T is generated by G if there is a sequence $u_0, \ldots, u_n$, for which:

0) $u_0 = A$;

1) for all $i = 0, \ldots, n-1$, there are some p, q, $(\varphi \rightarrow \psi) \in \Pi$ for which

$$u_i = p\varphi q\,, \quad u_{i+1} = p\psi q\,;$$

2) $u_n = w$.

As we know, Church's thesis is important for the philosophy of mathematics and mathematical practice, as well as for the philosophy of computer science. The same can be stated for the following thesis (and it is an interesting question in what sense it is equivalent to Church's thesis).

Post thesis. Each notion of provability (or enumerability or generability) is equivalent to the notion of generability by some grammar.

So, to characterize the provability in our theory, we have to find a proper grammar. We now try to express the notion of generated word for a particular grammar, G, in our language. First of all, we shall consider not words, but their codes; so we have to construct a formula $P(\cdot)$, which means that its argument is a code of the generated word. But now we must choose some method to code sequences of words (to say "there is a sequence"...). The straightforward way is to code $u_0, \ldots, u_n$ by $\Delta\tilde{u}_0\,\Delta\tilde{u}_1\,\Delta \ldots \Delta\tilde{u}_n\,\Delta$. Then we write three technical formulae:

$$\begin{aligned}
&Atom(y) && \rightleftharpoons \neg\,\exists p, q(y = p\Delta q)\\
&First(y, z) && \rightleftharpoons Atom(y)\ \&\ \exists p(\Delta y\Delta p = z)\\
&Last(y, z) && \rightleftharpoons Atom(y)\ \&\ \exists p(p\Delta y\Delta = z)\\
&Neib(s, t, z) && \rightleftharpoons Atom(s)\ \&\ Atom(t)\ \&\ \exists p, q(p\Delta s\Delta t\Delta q = z)\,.
\end{aligned}$$

Let us try to formalise a part of point 1) from the definition of generation in grammar G:

$$Step(u, v) \rightleftharpoons \bigvee_{(\varphi\to\psi)\in\Pi} \exists p, q(u = p\tilde{\varphi}q\ \&\ v = p\tilde{\psi}q)$$

Now, we obtain the formula describing the notion of provability:

$$P(x) \rightleftharpoons \exists z(First(\tilde{A}, z)\ \&\ \forall u, v(Neib(u, v, z) \to Step(u, v))\ \&\ Last(x, z))$$

Evidently, $P(x)$ means that x is the code of a provable word. And now, as in Goedel's proof:

$$D(x) \rightleftharpoons \neg\,P(S(x))\,.$$

g is the code of $D(x)$, in other words

$$g \rightleftharpoons \tilde{D}$$

and

$$G \rightleftharpoons D(g)$$

is a Goedel's formula.

To prove that, we have only to analyse, what $S(h)$ means. Let us return to the definition of S. We have:

Let h be the code of a formula φ with no bounded occurrences of variable x. Then $S(h)$ is the code of the formula $\varphi(h)$ (that is, of the formula obtained from φ by substituting h instead of each occurrence of x).

So G means that formula coded by $S(g)$ is unprovable. But this formula is $D(g) = G$ itself.

Goedel's Theorem is proved.

Our proof can be presented in an even more structured manner in the style of step-by-step development.

Operation S looks rather unnatural. In fact it can be easily expressed in our language without S. The idea here is the same as in construction of predicate P. The process of substitution is represented as a sequence of steps. But first of all we have to express the notion of coding in our language. This can be done similarly to the generation in a grammer also. Indeed; Z is the code of w if there exists a sequence $u_0 \ldots u_n$ for which

0) $u_0 = \Delta\Delta w$

1) $u_i = p\,\Delta\Delta\,0q$ and $u_{i+1} = p\,010\,\Delta\Delta q$ or
$u_i = p\,\Delta\Delta\,1q$ and $u_{i+1} = p\,0110\,\Delta\Delta q$

2) $u_n = z\,\Delta\Delta.$

As for S, the sequence of steps also appears in a natural way from the definition. Then our language becomes rather natural, as natural for the computer scientist and programmer as + (addition) and * (multiplication) is.

ACKNOWLEDGEMENT

I would like to express my gratitude to my Czech colleagues and friends, namely Dr. J. Bečvář and Dr. Z. Kratochvíl, who helped me to formulate clearly my ideas.

(Received July 6, 1987.)

REFERENCES

[1] В. М. Глушков: Теорема о неполноте формальных теорий с позиций программиста. Кибернетика (1979), №. 2, 1—5.

[2] А. Л. Семенов, В. А. Успенский: Математическая логика в вычислительных науках и в вычислительной практике. Вестник Академии наук СССР (1986), №. 7, 93—103.

[3] R. M. Smullyan: Theory of Formal Systems. (Annals of Mathematical Studies No. 47.) Princeton University Press, Princeton, N. J. 1962.

[4] В. А. Успенский: Теорема Геделя о неполноте. Наука, Москва 1982.

[5] В. А. Успенский: Теорема Геделя и теория алгоритмов. Успехи математических наук *8* (1953), №. 56, 176—178.

Alexei L. Semenov, Council for Cybernetics — Academy of Sciences, Vavilova 40, Moscow. U.S.S.R.