

**FACULTY
OF MATHEMATICS
AND PHYSICS**
Charles University

BACHELOR THESIS

Maroš Grego

**The incompleteness theorems and
Berry's paradox**

Department of Algebra

Supervisor of the bachelor thesis: prof. RNDr. Jan Krajíček, DrSc.

Study programme: Mathematics

Study branch: Mathematics for Information
Technologies

Prague 2022

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In date

Author's signature

I would like to thank my supervisor prof. RNDr. Jan Krajíček, DrSc. for prompt responses with useful comments and suggestions to this thesis.

Title: The incompleteness theorems and Berry's paradox

Author: Maroš Grego

Department: Department of Algebra

Supervisor: prof. RNDr. Jan Krajíček, DrSc., Department of Algebra

Abstract: This thesis is devoted to a formal presentation of an alternative proof of Gödel's first incompleteness theorem, based on the Berry paradox ("the smallest number not definable in under 57 characters", with this definition having less characters and defining this number). The approach used was suggested by an article by G. Chaitin. We define the Kolmogorov complexity of a natural number m as the binary length of the smallest program for the universal Turing machine that on input 0 outputs the number m . Using a formal argument based on the Berry paradox, we show that the property of a (large enough) number n being a lower bound for the Kolmogorov complexity of a number m is not provable in any consistent recursively axiomatizable extension of Robinson arithmetic. But by a counting argument, for all n , it is true for all but finitely many m . This is used to prove the first incompleteness theorem. Another way (by G. S. Boolos) of formalizing the Berry paradox to prove the same theorem is put in the context of the presented approach.

Keywords: first-order theories, incompleteness theorems and Berry's paradox

Contents

Introduction	2
1 Turing machines	3
1.1 Recursive and recursively enumerable languages	4
2 First-order theories and deductive systems	5
2.1 Deductive systems	5
2.2 Robinson arithmetic (Q)	6
2.3 Recursively axiomatizable extensions of Q	6
3 Turing-computable functions	7
3.1 Representability in Q	7
3.2 Universal Turing machine	7
4 Formalizing the Berry paradox	9
4.1 Kolmogorov complexity	9
4.2 Gödel numbers	9
4.3 Incompleteness	11
4.4 Relation to formula length	12
Conclusion	15
Bibliography	16

Introduction

In his original proof of his incompleteness theorem, Gödel [1931] formalized a variant of the so called liar's paradox, which is the sentence: "This sentence is not true." Roughly speaking, he constructed a sentence G , which could be interpreted as saying that the sentence G is not provable.

There are other ways to prove Gödel's result. In particular, as was observed by Vopěnka [1966] (for the second incompleteness theorem) and later by Chaitin [1971], another paradox may be used. Consider all strings of alphabetic characters (with spaces). Some of them can be interpreted as English sentences defining a unique natural number, such as "the smallest odd prime number", which defines the number 3. As there are finitely many strings of characters of, say, length n , they can define only finitely many natural numbers. Since the amount of natural numbers is infinite, there are numbers not definable in English in n characters, and there is the smallest one of them. Talking about it, we obtain a paradox, originally attributed by Russell [1908] to a librarian G. G. Berry:

Berry paradox. *The smallest number not definable in under 57 characters*

But this very definition uses 56 characters and defines that number!

The goal of this thesis is to present the Chaitin's argument formally. His idea is to consider the state complexity of Turing machines. After introducing prerequisites, which are mainly the properties of Turing machines and Robinson arithmetic, the Kolmogorov complexity of a number m is defined (as the smallest length of a program for the universal Turing machine which on input 0 outputs the number m). Then, it is shown that the property of Kolmogorov complexity of m being larger than n is expressible in the language of arithmetic, but not provable for a large enough n in a consistent theory. For otherwise, we could construct a program that, given a number n , would look through all provable theorems until it found this one, and output the number m . If n is sufficiently large, the length of this program is smaller than n - and yet, it outputs m . A Berry-style paradox.

Boolos [1989] suggested a possibly more natural way of making the Berry paradox formal, by talking about the length of the formulas that name a number m (in a sense that it is provable that the formula is true only for that particular m). In the final section, it is shown that the minimal length of a formula naming m is a form of Kolmogorov complexity and this approach of Boolos can be considered a special case of Chaitin's.

1. Turing machines

A Turing machine is a way to formalize the notions of computation and formal deduction. It might be pictured as a machine that performs its actions on one-dimensional tape, infinite in both directions. Furthermore, the machine is always in one of finitely many states and there is an arrowhead, inspecting one symbol of the tape at a time.

Definition 1.1. A Turing machine is a tuple (Q, Σ, δ) , where Q is a non-empty finite set called states, Σ is a non-empty finite set called an alphabet and

$$\delta : Q \times (\Sigma \cup \{\sqcup\}) \rightarrow Q \times (\Sigma \cup \{\sqcup\}) \times \{\leftarrow, \rightarrow\}$$

is called a transition function, where $\sqcup \notin \Sigma$ is a symbol called blank.

This means, when the machine is in a state $q \in Q$, the arrowhead is currently on a symbol σ and $\delta(q, \sigma) = (q', \sigma', D)$, it should enter the state q' , write the symbol σ' on the current cell and move the arrowhead one cell in the direction of D (left for \leftarrow or right for \rightarrow).

We pick a state $q_{\text{init}} \in Q$ called initial state, a subset $Q_{\text{halt}} \subset Q$, $q_{\text{init}} \notin Q_{\text{halt}}$ called halting states and we suppose at the beginning of computation, the machine is in the state q_{init} , only finitely many symbols on the tape are not blanks and the arrowhead starts at the leftmost non-blank symbol. Then, at each step of the computation, the state and the symbol are updated and the arrowhead moved according to the δ transition function. Reaching a state in Q_{halt} means the computation is finished (it is possible, though, that the machine never reaches such a state). We then say the machine halts.

Definition 1.2. Let Σ^* be a set of all strings of characters from Σ of finite length.

For $\alpha_1, \dots, \alpha_k, \omega \in \Sigma^*$, we say ω is the output of a Turing machine M on input $\alpha_1, \dots, \alpha_k$ if when M begins by scanning the leftmost character of $\alpha_1, \dots, \alpha_k$, separated by one blank symbol on an otherwise blank tape (every other character is blank), after finite time, M halts with ω on consecutive tape cells, with the head scanning the leftmost character of ω and blank everywhere else.

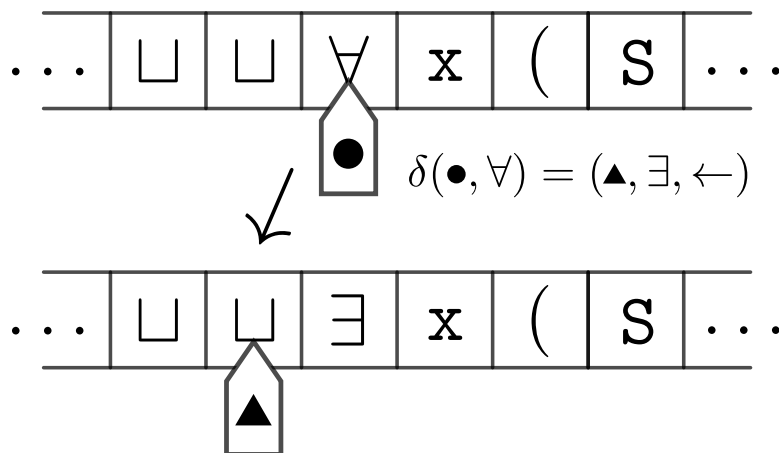


Figure 1.1: A state transition of Turing machine.

1.1 Recursive and recursively enumerable languages

A function is called *partial* if it may not be defined for every value in its domain.

A partial function $f : (\Sigma^*)^k \rightarrow \Sigma^*$ is called *partial recursive* if it is computed by a Turing machine (which on input $\alpha_1, \dots, \alpha_k$ outputs $f(\alpha_1, \dots, \alpha_k)$ if it is defined). f is called *total recursive* if it is defined everywhere.

A set $L \subseteq \Sigma^*$ is called *recursive* (or decidable) if its characteristic function is a total recursive function. L is called *recursively enumerable* if it is a range of partial recursive function (or empty).

Equivalently, L is recursively enumerable iff it is a range of a total recursive function.

Another useful characterization of a recursively enumerable set $L \subseteq \Sigma^*$ is as one for which there exists a Turing machine $M = (Q, \Sigma, \delta)$ and a state $q_{\text{accept}} \in Q_{\text{halt}}$ such that M halts in the state q_{accept} precisely for the inputs from L .

The proof that these definitions of recursively enumerable sets are equivalent can be found in Boolos et al. [2007, Section 8.3].

2. First-order theories and deductive systems

In what follows, we use first order logic with equality, with symbols \forall and \exists for universal and existential quantifier, \wedge and \vee for conjunction and disjunction, \rightarrow for implication, \leftrightarrow for equivalence and \neg for negation.

We will denote variables by lowercase Latin letters (x, y, \dots) and formulas by lowercase Greek letters (ϕ, ψ, \dots). We will consider only finite languages. (We can replace the variables x, y, z, \dots by x, x', x'', \dots , using a new symbol $'$, so that the used alphabet is finite.)

Formally, the logical sentences are strings of symbols that can be subject to mechanical inspection or manipulation, for example by a Turing machine, which uses the given language.

Well-formed sentences of a first order language (as defined e.g. in Boolos et al. [2007, Chapter 9]) are decidable. By the definition of a well-formed sentence, for each logical symbol, there is a clear syntactical formation rule (e.g. $(\phi) \wedge (\psi)$ is well-formed if ϕ and ψ are - for simplicity, we may assume the parentheses are never omitted). Given each such symbol, the machine checks whether the subformulas are well-formed and whether the formation rule is applied correctly. It will halt after finite time.

2.1 Deductive systems

We shall assume that a predicate calculus (a formal deductive system) is fixed that is based on a finite number of schematic inference rules and axiom schemes. A known consequence is that the set of valid proofs in this system, using a recursive set of non-logical axioms, is recursively enumerable.

Example. Modus ponens is a rule, which from the sentences $\phi \rightarrow \psi$ and ϕ produces the sentence ψ .

Example (axiom schema of universal instantiation). For every formula $\phi(x)$ with one free variable and any term t free for x in ϕ , there is an axiom

$$\forall x\phi(x) \rightarrow \phi(t)$$

.

We shall further assume that the deductive system satisfies the *Completeness theorem*: a sentence ϕ is a logical consequence of a set of sentences S (i.e. it is true in every model of S) iff it can be proven from S in this system (for which we will write $S \vdash \phi$). The fact that such systems exist was proven by Gödel [1930].

For a language L , an L -theory T is a set of sentences in L (called the theorems of T) closed under logical consequence. A theory is called *recursively axiomatizable* if there is a recursive set $S \subseteq T$, called the *axioms* of T , having the same logical consequences as T . By the remark above, any recursively axiomatizable theory is recursively enumerable (in fact, the opposite also holds).

2.2 Robinson arithmetic (Q)

Language of arithmetic is a first-order language L with a constant symbol 0 , an unary function symbol S and binary function symbols $+$ and \cdot , written in infix notation. S has priority over the other functions; where no ambiguity arises, parentheses will be dropped.

The intended domain of interpretation of this language is arithmetic, i.e. \mathbb{N} (the set of natural numbers with 0), with S representing the successor operation ($Sn = n+1$) and $+$, \cdot representing addition and multiplication of natural numbers.

Definition 2.1. *Robinson arithmetic or Q [Robinson, 1950] is a theory in the language of a arithmetic L consisting of the logical consequences of the following axioms (free variables are bound by implicit universal quantifiers):*

1. $Sx = Sy \rightarrow x = y$ (numbers with the same successor are the same)
2. $\neg(Sx = 0)$ (the successor of every number is not 0)
3. $x = 0 \vee \exists y(Sy = x)$ (a number is either 0 or a successor)
4. $x + 0 = x$
5. $x + Sy = S(x + y)$ (recursive definition of addition)
6. $x \cdot 0 = 0$
7. $x \cdot Sy = (x \cdot y) + x$ (recursive definition of multiplication)

2.3 Recursively axiomatizable extensions of Q

It is clear that the axioms of Q , when interpreted in arithmetic, are true. On the other hand, on its own, Q is quite a weak theory - even elementary theorems like $x + y = y + x$ are not provable in it [Boolos et al., 2007, Chapter 14]. For a recursively axiomatizable theory to contain more true statements about \mathbb{N} , more axioms need to be added. We say a theory T' is an extension of a theory T if $T \subseteq T'$. A famous example is:

Definition 2.2. *Peano arithmetic is a theory consisting of the consequences of the Peano axioms, which are the axioms of Q , along with, for each $k \in \mathbb{N}$, for each formula ϕ with $k + 1$ free variables x, y_1, \dots, y_k in the language of arithmetic L , the sentence*

$$(\phi(0, \bar{y}) \wedge \forall x(\phi(x, \bar{y}) \rightarrow \phi(Sx, \bar{y}))) \rightarrow \forall x\phi(x, \bar{y})$$

where \bar{y} denotes the tuple of variables y_1, \dots, y_k , which are implicitly universally quantified.

Thus the axioms of Peano arithmetic are those of Q and the axiom schema of induction - a new axiom for each formula of the language. That means there are infinitely many axioms, but since well-formedness of a formula and the sentence being an instance of this induction schema are decidable properties, it is recursively axiomatizable.

For what follows, we may consider any theory T which is a recursively axiomatizable extension of Q .

3. Turing-computable functions

For $k \in \mathbb{N}$, a partial function $f : \mathbb{N}^k \rightarrow \mathbb{N}$ is called *Turing computable* if there exists a Turing machine $(Q, \{0, 1\}, \delta)$ which, when given as input numbers n_1, \dots, n_k , for which f is defined, written in binary, outputs $f(n_1, \dots, n_k)$, written in binary, and otherwise does not halt.

The requirement of the binary alphabet $\{0, 1\}$ is not necessary - any k -ary would do. The symbols of the binary alphabet will be called *bits*.

A formal definition and some equivalent characterisations of Turing computable functions are given in Boolos et al. [2007, Chapter 4–6]. In fact, any known effective way of computing functions of natural numbers gives a subclass or an equal class of functions to Turing computable functions [Boolos et al., 2007, Chapter 6]. The informal statement that it must always be the case is known as the *Church-Turing thesis*. An important example is the class of functions representable in Robinson arithmetic.

Example. Let $\ell(n)$ denote the length of $n \in \mathbb{N}$ when written in binary. It is Turing computable - a Turing machine can keep a counter next to the input, replace the input digits by blanks and for each, increase the counter by one.

Example. For f, g Turing computable functions, their composition $f \circ g$ is Turing computable. Indeed, a run of a machine computing g can be followed by a run of a machine computing f on the output of g .

3.1 Representability in Q

For $n \in \mathbb{N}$, let \mathbf{n} (in boldface) be the term $\underbrace{SS \dots S}_n 0$, called the *numeral* of n .

Definition 3.1. A partial function $f : \mathbb{N}^k \rightarrow \mathbb{N}$ is said to be representable in the theory T (in the language of arithmetic L) if there is a formula ϕ with $k + 1$ free variables y_1, \dots, y_k, x which holds only for the graph of f , i.e. for $n_1, \dots, n_k \in \mathbb{N}$, $f(n_1, \dots, n_k)$ is defined iff

$$T \vdash \forall x (\phi(\mathbf{n}_1, \dots, \mathbf{n}_k, x) \leftrightarrow x = \mathbf{m})$$

where $m = f(n_1, \dots, n_k)$.

Theorem 3.2. Every Turing computable function is representable in the Robinson arithmetic Q .

Proof. The proof can be found in Boolos et al. [2007, Chapter 16]. □

3.2 Universal Turing machine

For a two place function f , let $f(m, -)$ denote the function $n \mapsto f(m, n)$, i.e. the function obtained from f by "plugging in" m in place of the first variable.

Theorem 3.3 (Universal Turing machine). *There exists a Turing computable function $u : \mathbb{N}^2 \rightarrow \mathbb{N}$ such that for every Turing computable function $f : \mathbb{N} \rightarrow \mathbb{N}$, there is a natural number $\ulcorner f \urcorner$, called a program for f , for which $u(\ulcorner f \urcorner, -) = f$ (equal as partial functions $\mathbb{N} \rightarrow \mathbb{N}$).*

Moreover, such an u can be constructed with following properties:

1. $\ell(\ulcorner f \circ g \urcorner) = \ell(\ulcorner f \urcorner) + \ell(\ulcorner g \urcorner) + c$ with c constant, not depending on f or g ,
2. for the function c_n returning n on 0 and undefined elsewhere, $\ell(\ulcorner c_n \urcorner) = O(\ell(n))$,

where $\ell(n)$ is the binary length of the number n .

Proof. One of the ways of constructing the universal Turing machine is presented in Boolos et al. [2007, Chapter 8]. When M is a Turing machine computing f , $\ulcorner f \urcorner$ can be considered as an encoded description of states and the transition function of M that the universal Turing machine U computing u can follow.

From the construction follows the property (1), since the program for $f \circ g$ consists of encoded transition function for a machine computing f , followed by the same for g .

The construction also implies the property (2), as c_n can be computed by a machine with $O(\ell(n))$ states that simply writes the bits of n on the output. □

Although, strictly speaking, *universal Turing machine* is a name for the Turing machine U that computes u , for brevity, we will also use this name to refer to the function u itself.

4. Formalizing the Berry paradox

4.1 Kolmogorov complexity

Let $x \leq y$ denote the formula $\exists z(z + x = y)$, meaning in \mathbb{N} that x is smaller than or equal to y .

Let u be the universal Turing machine from the theorem 3.3 and let τ be the formula that represents it in Q . Let λ be the formula that represents ℓ (the binary length of the number). Finally, let κ be the formula

$$\kappa(x, y) = \forall p \forall l (\tau(p, 0, x) \wedge \lambda(p, l) \rightarrow y \leq l)$$

Interpreting it in \mathbb{N} , $\kappa(\mathbf{m}, \mathbf{n})$ holds when every program for the universal Turing machine u that on input 0 outputs the value m (written in binary) is of length at least n . We define the *Kolmogorov complexity* $K(m)$ as the smallest n such that $\kappa(\mathbf{m}, \mathbf{n})$ (i.e. the length of the smallest program that outputs m on the input 0).

Note that the definition of $K(m)$ includes a choice of a particular universal Turing machine u .

Since there are 2^n programs of length smaller than n and each produces at most one number on the input 0, there are only finitely many m such that $K(m) < n$. There are infinitely many natural numbers, so for every n , all but finitely many m satisfy $K(m) \geq n$.

We will show that in Q , or any consistent sound recursively axiomatizable extension of it, for a large enough n (dependent on the theory), this is not provable.

4.2 Gödel numbers

Any finite alphabet Σ can be encoded using the binary alphabet $\{0, 1\}$. One of the simplest ways to do that is to assign a unique l -bit sequence to each element of Σ , where $2^l \geq |\Sigma|$. For example, the language of arithmetic could be encoded as follows (recall that instead of variables x, y, z, \dots , we can formally use x, x', x'', \dots , using two symbols x and $'$, so that the language is finite):

0	+	(\forall	\wedge	\rightarrow	\neg	x
0000	0010	0100	0110	1000	1010	1100	1110
S	\cdot)	\exists	\vee	\leftrightarrow	=	'
0001	0011	0101	0111	1001	1011	1101	1111

Any such encoding e can be extended to an encoding $e^* : \Sigma^* \rightarrow \{0, 1\}^*$ simply by concatenating the encoded words, so for example

$$e^*(\forall x(x = x)) = 0110 \ 1110 \ 0100 \ 1110 \ 1101 \ 1110 \ 0101$$

The particular choice of codes is not essential. For the following proof, it matters that the code of every character has the same bit length. However, more sophisticated encodings can be created where this doesn't have to hold.

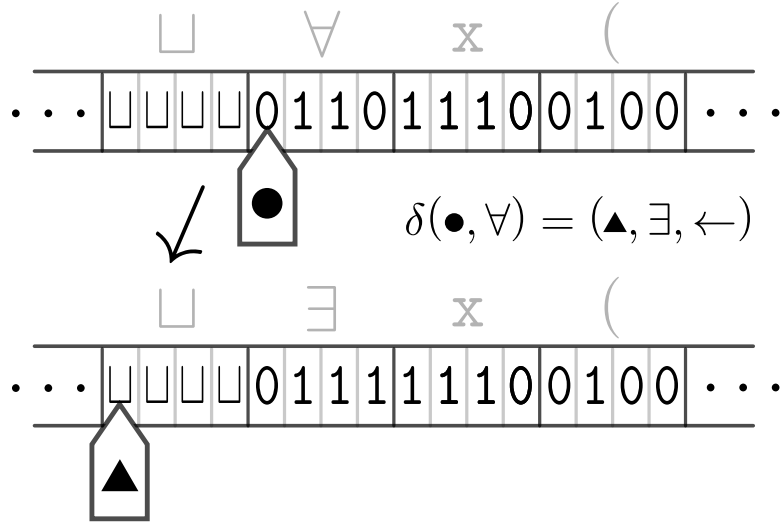


Figure 4.1: A sequence of state transitions of the Turing machine M_e , where instead of each original symbol, there is a block of l 0s and 1s (or \sqcup s).

Theorem 4.1. *For any injective encoding $e : \Sigma \rightarrow \{0, 1\}^l$ and any Turing machine $M = (Q, \Sigma, \delta)$, there is a Turing machine $M_e = (Q_e, \{0, 1\}, \delta_e)$ that computes the same (partial) function encoded via e . Explicitly, M halts on input $\alpha_1, \dots, \alpha_k$ with output ω iff M_e halts on input $e^*(\alpha_1), \dots, e^*(\alpha_k)$ with output $e^*(\omega)$.*

Proof. In the beginning, M_e replaces each blank symbol separating the input strings by l consecutive blank symbols and moves the arrowhead back to its starting position (the leftmost non-blank symbol).

Then, action of the machine M_e can be thought of as looking at l consecutive cells from left to right and then having a transition based on their content. Specifically, Q_e contains every state $q \in Q$. For each of them, on reading the symbol 0 or 1, it reads the symbols from $l - 1$ further cells, eventually going into the state q_γ , where $\gamma \in \{0, 1\}^l$ is the word it has read. If there is $\sigma \in \Sigma$ such that $e(\sigma) = \gamma$ and $\delta(\sigma, q) = (\sigma', q', D)$, the machine writes on these l cells $e(\sigma')$ (or l blanks if $\sigma' = \sqcup$), moves onto the first cell of the nearby block of l cells in the direction D and enters the state q' .

On blank, if the next l symbols are blanks, it acts analogously - if $\delta(\sqcup, q) = (\sigma', q', D)$, it writes $e(\sigma')$ instead of the blanks (or keeps the blanks if $\sigma' = \sqcup$), moves in the direction D and enters the state q' .

If the input is not in valid format (it is not an encoding of anything), it enters some state where it loops forever. □

With an encoding, any recursive function on the string of symbols of some finite alphabet can be thought of as a computable function on numbers. For a binary encoding e and $\omega \in \Sigma^*$, let $e^b(\omega)$ be $1e^*(\omega)$ (the symbol 1 concatenated with $e^*(\omega)$). It can be considered as a binary representation of a natural number - we will denote this number $\lceil \omega \rceil$. This way, the encoding assigns a unique natural number to every string of symbols of the alphabet.

Corollary 4.2. *For any injective encoding $e : \Sigma \rightarrow \{0, 1\}^l$ and any Turing machine $M = (\Sigma, Q, \delta)$, there is a Turing machine M_b that computes the same (partial) function in binary on natural numbers encoded via e^b . Explicitly, M halts on input $\alpha_1, \dots, \alpha_k$ with output ω iff M_b halts on input $e^b(\alpha_1), \dots, e^b(\alpha_k)$ with output $e^b(\omega)$.*

In other words, if M expects k strings on the input and outputs one string, the corresponding (partial) function on these strings encoded via e^b is a Turing-computable function on natural numbers.

Proof. M_b erases 1 from the beginning of each number, moves the arrowhead to the leftmost non-blank symbol and runs the machine M_e from the previous theorem. After M_e halts, it writes 1 at the beginning of the output. □

So rules of inference and other recursive transformations of strings in the language of arithmetic, with an appropriate encoding, correspond to Turing computable functions on natural numbers (which encode such strings). Since this idea appeared in the original Gödel's proof, the numbers assigned to formulas by such encodings are usually called *Gödel numbers*.

4.3 Incompleteness

Definition 4.3. *A theory is called consistent if contains no contradiction (it does not contain a sentence and its negation).*

The idea of the following proof is due to Chaitin [1971].

Theorem 4.4. *For T consistent recursively axiomatizable extension of Q , for any sufficiently large $n \in \mathbb{N}$ and any $m \in \mathbb{N}$, the sentence $\kappa(\mathbf{m}, \mathbf{n})$ is not a theorem of T .*

Proof. Suppose $T \vdash \kappa(\mathbf{m}, \mathbf{n})$. If the Kolmogorov complexity $K(m)$ of m is smaller than n , meaning there is a program p for the universal Turing machine u of binary length smaller than n that on input 0 outputs m , then

$$Q \vdash \tau(\mathbf{p}, 0, \mathbf{m}) \wedge \exists l(\lambda(\mathbf{p}, l) \wedge \neg(\mathbf{n} \leq l))$$

(since τ represents u and λ represents the binary length), making T (an extension of Q) prove a contradiction with $\kappa(\mathbf{m}, \mathbf{n})$. So we may further assume there is no such p .

Since the theory T is recursively axiomatizable, it is recursively enumerable, i.e. it is a range of a recursive function, which we can assume to be total. Let E be a Turing machine computing this function.

Let M be a Turing machine that expects some numeral \mathbf{k} on the input, then checks all the outputs of E , until it finds one of the form $\kappa(\mathbf{m}, \mathbf{k})$ for some numeral \mathbf{m} and outputs \mathbf{m} . By assumption, in case $k = n$, such a proof is found after finite time.

Let M_b be the machine from the corollary 4.2, working as M , but with the strings encoded as natural numbers in binary using some encoding. Let M'_b be a machine which gets some natural number k on the input in binary, rewrites it

into $e^b(\mathbf{k})$, runs M_b on it, which eventually outputs an encoded numeral $e^b(\mathbf{m})$, and then rewrites it into the number m in binary and halts.

M'_b computes a function $f : \mathbb{N} \rightarrow \mathbb{N}$. So there is a program $\ulcorner f \urcorner$ for the universal Turing machine u such that $u(\ulcorner f \urcorner, -) = f$.

Let M_n be a machine, which on input 0 rewrites the content of the tape to n (in binary), goes to the beginning of it and runs M'_b . It computes a function $f_n : \mathbb{N} \rightarrow \mathbb{N}$. By the properties of u ,

$$\ell(\ulcorner f_n \urcorner) = O(\ell(n) + \ell(\ulcorner f \urcorner)) = O(\log_2 n)$$

($\ell(\ulcorner f \urcorner)$ does not depend on n). Therefore $\ell(\ulcorner f_n \urcorner)/n \rightarrow 0$ as $n \rightarrow \infty$ (linear growth eventually surpasses logarithmic growth), so if n is sufficiently large, $n > \ell(\ulcorner f_n \urcorner)$. Because M_n outputs the number m on the input 0, $K(m) \leq \ell(\ulcorner f_n \urcorner) < n$, meaning the Kolmogorov complexity of m is smaller than n , yielding a contradiction. □

Corollary 4.5. *The true arithmetic, i.e. the set of sentences true in the standard model of arithmetic (assuming it is consistent) is not recursively axiomatizable*

Proof. Arithmetic is a consistent extension of the Robinson arithmetic Q . By a counting argument as in the section 4.1, for any n , there is m of Kolmogorov complexity larger than n , so $\kappa(\mathbf{m}, \mathbf{n})$ is true in arithmetic. But that means arithmetic can not be recursively axiomatizable. □

Definition 4.6. *A theory is called complete if for every sentence ϕ in its language, it either contains ϕ or $\neg\phi$.*

A theory in the language of arithmetic L is called sound if all of its theorems are true in the standard interpretation of the language (\mathbb{N} with zero, successor, addition and multiplication).

Corollary 4.7. *No consistent sound recursively axiomatizable extension of Q is complete.*

Proof. For m, n with n so large that the theorem 4.4 holds and $\kappa(\mathbf{m}, \mathbf{n})$ is true in arithmetic, neither this sentence nor its negation can be a theorem of a consistent sound extension of Q . □

4.4 Relation to formula length

Boolos [1989] suggested using the Berry paradox to prove the incompleteness theorem in a slightly different way, more akin to the original formulation of the paradox. Namely, he considered a formula to name a number if it is provable that it is true only for that number.

Definition 4.8. A formula ψ with one free variable x (in the language of arithmetic L) is said to name a number $m \in \mathbb{N}$ in the theory T if

$$T \vdash \forall x(\psi(x) \leftrightarrow x = \mathbf{m}))$$

We can see it is a special case of definition 3.1, where we consider the constant m as a nullary function $\mathbb{N}^0 \rightarrow \mathbb{N}$.

Boolos then proceeded to hint at a way of formalizing the notion of the smallest number not nameable by a formula by less than n symbols. But alas, for n sufficiently large, the formula naming it has less than n symbols!

Kikuchi et al. [2012] noted that this proof is essentially a special case of Chaitin's proof, in the following sense. Consider a Turing machine P , working in the language of arithmetic, expecting on the input a formula ϕ with two free variables y, x and a numeral \mathbf{n} . It then uses the machine E from the proof of the theorem 4.4, searching through all the theorems of T until it finds one that proves

$$\forall x(\phi(\mathbf{n}, x) \leftrightarrow x = \mathbf{m}))$$

for some numeral \mathbf{m} . It then outputs \mathbf{m} .

By the corollary 4.2, we can construct a Turing machine P_b that works as P , but on strings of symbols encoded as natural numbers. Finally, let P'_b be a machine that gets numbers p, n on the input, replaces n with $e^b(\mathbf{n})$ (encoding of the numeral of n), runs P_b on this input, possibly obtaining $e^b(\mathbf{m})$ for some numeral \mathbf{m} on the output and then replaces it with m , written in binary.

Claim 4.9. For T consistent recursively axiomatizable extension of Q , P'_b is a universal Turing machine, which satisfies the properties from the theorem 3.3.

Proof. By the theorem 3.2, every Turing computable function f is representable in Q by a formula ϕ , so if f is defined on n , on input $\phi(x, y), \mathbf{n}$, after finite time, P reaches the formula $\forall x(\phi(\mathbf{n}, x) \leftrightarrow x = \mathbf{m})$ and thus P'_b on input $\ulcorner \phi(x, y) \urcorner, n$ outputs m . If such formula was found for some $m \neq f(n)$, T would be inconsistent. So $\ulcorner \phi(x, y) \urcorner$ is a program for f .

Note that the binary length of the number $\ulcorner \phi(x, y) \urcorner$ is a constant multiple of the length of the string $\phi(x, y)$ (in the language of arithmetic) plus 1, since every character is encoded by the same number of bits.

For the property (1), if ϕ represents f and ψ represents g , the formula

$$\forall z(\psi(x, z) \rightarrow \phi(z, y))$$

represents $f \circ g$ and has the length of the sum of lengths of ϕ and ψ plus constant.

For the property (2), for a number n of the form $1\mathbf{b}_1\mathbf{b}_2 \dots \mathbf{b}_k$ when written in binary, the formula

$$x = 0 \wedge y = \mathbf{b}_k + 2 \cdot (\mathbf{b}_{k-1} + 2 \cdot (\mathbf{b}_{k-2} + 2 \cdot (\dots)))$$

represents the function c_n (defined as n on 0 and undefined elsewhere), where n is expressed using Horner's rule, so the length of this formula is $O(\ell(n))$. □

Therefore the minimal length of a naming formula is a form of Kolmogorov complexity. Since the proof of the theorem 4.4 was agnostic to the choice of the universal Turing machine, this one, for which the programs are (encoded) formulas that represent functions, works as well. Explicitly, the sentence $\kappa(\mathbf{m}, \mathbf{n})$, defined using this universal Turing machine, means that every formula that "names" the number m (in the sense that $\phi(0, x)$ is provable only for m in the given theory) has length at least n , when encoded via a particular binary encoding (so the formula must use at least $\frac{n-1}{4}$ symbols when the encoding of section 4.2 is used). For n large enough, this sentence is not provable in any consistent extension of Q .

There is a minor apparent discrepancy with Boolos' approach, in that he considers a formula $\psi(x)$ with one free variable to name the number m if it is provably true only for m , while here, we consider a formula $\phi(y, x)$ with two free variables to "compute" m if $\phi(0, x)$ is provably true only for m . However, a formula of the form $\psi(x)$ can be interpreted as the one of the form $\phi(0, x)$ and vice versa (we do not require all free variables to be used in the formula), so this difference is not any essential.

Conclusion

Firstly, we defined the concepts of a Turing machine, recursive and recursively enumerable sets, the notion of a deductive system for the first order logic and its fundamental properties. Then, the axioms of Robinson arithmetic Q were presented.

In the third chapter, the notion of Turing-computable functions of natural numbers was defined. Their characterisation as functions representable in Q was stated. Then, the universal Turing machine was introduced, along with some natural requirements for the binary length of its programs.

Finally, we defined the Kolmogorov complexity of a natural number, along with the formula $\kappa(x, y)$, expressing that y is a lower bound for the Kolmogorov complexity of x (using a formula representing the universal Turing machine). A particular way of encoding formulas of the language of arithmetic was presented. It was proven that to any transformation of strings computable by a Turing machine corresponds a Turing computable function on natural numbers, which encode those strings.

Using the encoding, a formal argument using the Berry paradox was made. For n large enough and m such that $\kappa(\mathbf{m}, \mathbf{n})$ is true in arithmetic, if this sentence would be provable, then a Turing machine can be constructed that searches through all the theorems of the given theory until it finds it and outputs the number m . Since the binary length of the program of this machine is asymptotically logarithmic, with n large, it is smaller than n , contradicting $\kappa(\mathbf{m}, \mathbf{n})$.

In the ultimate section, the suggestion of Boolos to formalize the Berry paradox using the formula length was considered in the context of the presented approach. It was shown that a particular universal Turing machine can be constructed, which takes encoded formulas of arithmetic as its programs. For this machine (and the formula that represents it in Q), we can define the corresponding formula κ , used in the herein presented proof.

Bibliography

- G. S. Boolos. A new proof of the Gödel incompleteness theorem. *Notices of American Mathematical Society*, 36(4):388–390, April 1989.
- G. S. Boolos, J. P. Burgess, and R. C. Jeffrey. *Computability and Logic*. Cambridge University Press, 5 edition, 2007. doi: 10.10MathematicalLogic17/CBO9780511804076.
- G. J. Chaitin. Computational complexity and Gödel’s incompleteness theorem. *SIGACT News*, (9):11–12, apr 1971. ISSN 0163-5700. doi: 10.1145/1247066.1247068.
- K. Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematik und Physik*, 37:349–360, 1930. ISSN 0026-9255. doi: 10.1007/BF01696781.
- K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931. ISSN 0026-9255. doi: 10.1007/BF01700692.
- M. Kikuchi, T. Kurahashi, and H. Sakai. On proofs of the incompleteness theorems based on Berry’s paradox by Vopěnka, Chaitin, and Boolos. *Mathematical Logic Quarterly*, 58(4-5):307–316, 2012. doi: 10.1002/malq.201110067.
- R. M. Robinson. An essentially undecidable axiom system. *Proceedings of the International Congress of Mathematics*, 1:729—730, 1950.
- B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30(3):222–262, 1908. doi: 10.2307/2272708.
- P. Vopěnka. A new proof of the Gödel’s result on non-provability of consistency. *Bulletin de l’Academie Polonaise des Sciences*, 14:111–116, 1966.