NOTES ON HILBERT'S 10^{th} PROBLEM

MARCELLO MAMINO

ABSTRACT. These hastily written notes contain at least one mistake, and probably many more.

1. INTRODUCTION

We will assume *exponentiation is Diophantine* [Matiyasevich, chapter 2] and prove that Hilbert's tenth problem is undecidable. Our proof follows [Matiyasevich, chapters 3–5], however we will take shortcuts. All variables range over the non-negative numbers $\mathbb{N}$. Notationally, we will happily use x^y, $x < y$, etc. in arithmetic formulas, knowing that these relations are Diophantine. The notations $x \operatorname{div} y$ and $x \bmod y$ represent respectively the integer part of the quotient and the remainder. The word *sequence* here denotes a usually finite sequence $a_1 a_2 \dots a_n$ of usually Boolean (i.e. $0, 1$) symbols: we will often omit the words *Boolean* and *finite* when there is no risk of confusion, the set of finite sequences of elements of S is denoted $S^\star$. The word *tuple* is used to indicate a finite sequence in a context in which the length of that sequence is fixed (this distinction is admittedly a bit sophistic).

Strategy: First, we will establish a one-to-one mapping between positive numbers and finite Boolean sequences – the book develops instead codes for arbitrary sequences, but such codes are not strictly needed here. This allows us to identify numbers and sequences. Then we will prove that certain operations on sequences are Diophantine (the definition of Diophantine makes sense for sequences because – remember – we are identifying numbers and sequences through the 1-1 mapping, so, for us, a sequence is just a number). After this step we will forget the details of our coding of sequences into numbers and never use them again. Thanks to the Diophantine operations that we have constructed, we will be able to simulate by Diophantine conditions arbitrary Turing machines, thus proving that all recursively enumerable (r.e.) sets are Diophantine. This concludes the proof, because the halting set is r.e. but not decidable.

2. CODING OF SEQUENCES

We represent the sequence $a_1 a_2 \dots a_n \in \{0, 1\}^\star$ by the number $(1 a_n a_{n-1} \cdots a_1)_2$, where $(\spadesuit)_2$ denotes the value in binary notation, i.e.

$$(1 a_n a_{n-1} \cdots a_1)_2 \;=\; 2^n + \sum_{i=1}^{n} a_i \cdot 2^{i-1}$$

Date: 23·III·2017.

Notice that the digits in the number appear reversed with respect to the sequence (e.g. the sequence 111000 is represented by $(1000111)_2 = 71$), the reason is that it is convenient to take the unit's place as the first digit in a number. This convention is not a necessity: we could as well work with the opposite. On the contrary, the leading 1 that we add in front of the number *is* a necessity: omitting it would mean that the two different sequences 101 and 101000 get the same code 5.

Note (historical): the more one considers the order of places in our decimal system of writing numbers, the more it feels backwards. For instance, we write left-to-right, but we add and subtract numbers right-to-left. In fact, our decimal notation was introduced to Europe in 1202 by Fibonacci who learned it from the Arabs (that in turn obtained it from the Indians, in the VIII *century or thereabouts). Lo and behold: the Arabs wrote, and write, right-to-left!*

Note (technical): our addition of the leading 1 *might seem totally ad hoc. In fact, however, it is interesting to notice that all binary numbers with the only exception of* 0 *begin with a* 1*. It follows that, if we accept to special-case the number* 0*, then we can make each of the remaining numbers one digit shorter, by simply omitting that leading* 1 *which is always there! This idea actually finds a practical application in the IEEE 754 standard for the in-memory representation of floating point numbers, which is used in all modern (and also not so modern) computers. According to this standard, for instance, the number* $(1011.1011)_2 = 2^3(1.0111011)_2$ *is represented in single precision by the 32-bit word* $\underbrace{0}\,\underbrace{10000010}\,\underbrace{01110110000000000000000}$*. The first* 0 *represents the (positive) sign, the next eight bits represent the exponent* 3 *of* 2^3*, written in a funny way, the final 23 bits are the fractional part of* 1.0111011*, and the leading* 1 *is omitted. In conclusion we might say that we didn't add any leading* 1 *at all: that* 1 *would not be there in the first place, were the numbers written properly!*

A few operations on sequences are trivially Diophantine, let us list here below the most important ones.

1 - To represent any constant sequence, we can just use the corresponding numeric constant.

2 - The sequence $\overbrace{00\cdots0}^{n}$ of n consecutive 0s is 2^n.

3 - The length $\mathrm{len}(a)$ of a is the only x such that $2^x \leqslant a < 2^{x+1}$.

4 - The concatenation $a^\frown b$ representing the sequence $a_1 a_2 \dots a_{\mathrm{len}(a)} b_1 b_2 \dots b_{\mathrm{len}(b)}$ is $a + (b-1)\cdot 2^{\mathrm{len}(a)}$.

5 - The operations *shift left* and *shift right*, which we denote respectively by $a \ll n \stackrel{def}{=} a_{n+1} a_{n+2} \dots a_{\mathrm{len}(a)}$ and $a \gg n \stackrel{def}{=} \underbrace{00\cdots0}_{n} a_1 a_2 \dots a_{\mathrm{len}(a)}$ correspond to division and product by 2^n.

6 - The sequence $(x)_2$ representing the number x in binary (least significant digit first) is a Diophantine function of x, namely $(x)_2 = x + 2^\alpha$ where 2^α is the unique power of 2 such that $2^\alpha \operatorname{div} 2 \leqslant x < 2^\alpha$.

00101	00100
10100	10010
overl.	non overl.

overlap of sequences

Of course, to test two sequences for equality we only need to test the corresponding numbers for equality, because the mapping is bijective. There is a second test that we need to implement, and this will be the only non-trivial operation on sequences that we study, namely the test for non-overlap. We say that the sequences a and b of the same length are *non overlapping* if for all $i \leqslant \mathrm{len}(a) = \mathrm{len}(b)$ at most one of a_i and b_i is 1.

Lemma 1. *The relation "a and b are non overlapping" is Diophantine.*

Proof. Let us say that the numbers m and n are non overlapping if for all i at most one of the numbers $m \operatorname{div} 2^i$ and $n \operatorname{div} 2^i$ is odd. In other words, m and n are non overlapping if and only if, when written in binary, one below the other with the digits aligned to the right as for computing the sum, each column contains at most one 1. Clearly the sequences a and b are non overlapping if and only if $a - 2^{\mathrm{len}(a)}$ and $b - 2^{\mathrm{len}(b)}$ are non overlapping. So it remains to prove that the relation "the numbers m and n are non overlapping" is Diophantine.

101	100
10**1**00	10010
overl.	non overl.

overlap of numbers

We claim that m and n are non overlapping if and only if $\binom{m+n}{m}$ is odd. To prove this, let $v_2(x)$ denote the dyadic valuation of x, namely the largest y such that 2^y divides x. Then let $\#_1(x)$ denote the number of 1s in the binary representation of x. Observe that

$$v_2(x!) = x - \#_1(x)$$

This is easy to prove by induction, for instance using

$$\begin{aligned} v_2(x!) &= v_2(\,1\cdot 2\cdot 3\cdot 4\cdot 5\cdot 6\cdot 7\cdot 8\cdot 9\cdots x\,) \\ &= v_2(\,2\cdot 4\cdot 6\cdot 8\cdots 2(x\operatorname{div}2)\,) \\ &= v_2(\,2^{x\operatorname{div}2}\cdot(\,1\cdot 2\cdot 3\cdot 4\cdots x\operatorname{div}2)\,) \\ &= x\operatorname{div}2 + v_2((x\operatorname{div}2)!) \end{aligned}$$

Now, $\binom{m+n}{n}$ is odd if and only if its dyadic valuation vanishes, namely

$$\begin{aligned} 0 = v_2\left(\binom{m+n}{m}\right) &= v_2\left(\frac{(m+n)!}{m!n!}\right) \\ &= v_2((m+n)!) - v_2(m!) - v_2(n!) \\ &= \#_1(m) + \#_1(n) - \#_1(m+n) \end{aligned}$$

It is easy to see that this is the number of carries that occur when computing the sum $m+n$ in binary. In fact, each carry carries away precisely one 1. Clearly there are no carries if and only if m and n are non overlapping, we have thus established our claim. It remains to prove that the property "$\binom{m+n}{n}$ is odd" is Diophantine, to this end it suffices to prove that the function mapping x and y to $\binom{x}{y}$ is Diophantine.

By the binomial theorem

$$\begin{aligned} ((2^x+1)^x \operatorname{div} 2^{xy}) \bmod 2^x &= \left(\left(\sum_{i=0}^{x}\binom{x}{i}2^{xi}\right)\operatorname{div}2^{xy}\right)\bmod 2^x \\ &= \left(\left(\sum_{i=0}^{y-1}\binom{x}{i}2^{xi}\right)\operatorname{div}2^{xy}\right. \\ &\qquad \left. + \binom{x}{y} + \sum_{i=y+1}^{x}\binom{x}{i}2^{x(i-y)}\right)\bmod 2^x \end{aligned}$$

The first summand is nil because

$$\sum_{i=0}^{y-1}\binom{x}{i}2^{xi} \;<\; \left(\sum_{i=0}^{x}\binom{x}{i}\right)2^{x(y-1)} \;=\; 2^{xy}$$

The third summand is nil mod 2^x, therefore it remains

$$((2^x+1)^x \text{ div } 2^{xy}) \bmod 2^x = \binom{x}{y} \bmod 2^x = \binom{x}{y}$$

□

The proof of the previous lemma contains all the number theory that we will need in these notes. The next section will be devoted to a surprisingly powerful application of this innocent looking proposition.

3. Parallel computation

A function $f\colon \{0,1\}^n \to \{0,1\}$ can be said to induce a function $[f]$ from n-tuples of sequences of the same length l to sequences of length l, by componentwise action: let $a^1, a^2 \dots a^n$ be n such sequences, then

$$\left([f](a^1, a^2 \dots a^n)\right)_i \stackrel{def}{=} f(a^1_i, a^2_i \dots a^n_i) \tag{1}$$

for all $i = 1 \dots l$. The goal of this section is to prove that this function $[f]$ mapping a n-tuple of sequences to a sequence is Diophantine *in a way independent from* l. In other words there is a Diophantine function, that we denote $[f]$, such that for all l equation (1) holds.

Although the method will turn out to be quite simple, it is hard to overemphasize that this is the crux of these notes. Here, using a fixed Diophantine function, we prove that we can operate on all elements of the input sequences *in parallel*, independently of the length of these sequences. It is at this point that we gain the ability to represent with a single formula unboundedly many computations.

Lemma 2. *Let* f *be as above, then* $[f]$ *is Diophantine.*

Proof. Since every Boolean function can be written as a composition of negations and conjunctions, it suffices to prove that $[f]$ is Diophantine when f is either the function *not* or the function *and*.

Negation is immediate: $b = [not](a)$ if and only if $a + b = 3 \cdot 2^{\text{len}(a)} - 1$.

For the conjunction, first observe that, if x and y are non overlapping sequences of the same length, then $[or](x, y) = x + y - 2^{\text{len}(x)}$. Now, "$c = [and](a, b)$" is equivalent to "there are sequences x, y such that a, b, c, x, y have the same length, x, y, c are pairwise non overlapping, $a = [or](c, x)$, and $b = [or](c, y)$". □

This section concludes the part of the notes in which we investigate the Diophantine properties of our coding of sequences. Starting from the next section, we will only use sequences through operations that have been constructed up to here.

4. Coding of Turing machines

Starting from this section, we assume familiarity with Turing machines. Our goal is to find a *Diophantine representation* of arbitrary – any number of states and symbols – single tape Turing machines. Multi-tape Turing machines could be represented somewhat similarly, but we will not not need them. Our first goal is to represent

in a convenient way, by means of Boolean sequences, the configuration of such a Turing machine—namely the tape plus the internal state plus the position. Here we will assume that *the tape of our Turing machine has finite length* l notice that this contrasts with the usual convention of using a infinite or half-infinite tape.

First, we can assume that our machine has 2^m symbols, consecutively numbered from 0, and 2^n states, also consecutively numbered from 0. We will represent a configuration of the machine using $m+n+2$ sequences of length $l+2$.

1 - The sequence *fmt* is just

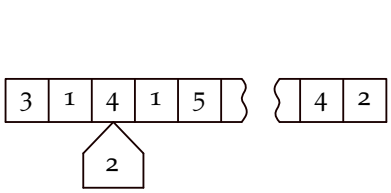
$$fmt \stackrel{def}{=} 1\overbrace{0\,0\cdots 0}^{l}1$$

the operation of the machine will not alter this sequence, we need it for merely technical reasons, to mark the beginning and the end of the tape. Observe that *fmt* can be produced as a Diophantine function of l, because it is just a concatenation of (twice) the constant sequence 1 and a sequence of l repeated 0s. For us, the places 1 and $l+2$ – those marked by a 1 in *fmt* – are not part of the tape: the other sequences will always contain 0s at this places. The l cells of the tape correspond to positions $2 \ldots l+1$.

2 - Then we have m sequences $tp^1, tp^2 \ldots tp^m$ representing the tape. The fact that symbol x is at place i in the tape is represented writing the m bits of the binary representation of x at places $tp^1_{i+1}, tp^2_{i+1} \ldots tp^m_{i+1}$ of these sequences. If the machine starts with an empty tape, then we can say that every cell of the tape contains the symbol 0, and the sequences $tp^1, tp^2 \ldots tp^m$ are each initialized to $l+2$ repeated 0s.

3 - We have one single sequence *pos* indicating the position of the machine. This sequence contains one single 1 at the spot occupied by the machine. It is initialized to 01 followed by l many 0s.

4 - Finally, we have n sequences $st^1, st^2 \ldots st^n$ that keep the state of the machine. When the machine is at position i, these sequences contain the binary representation of the state of the machine in $st^1_{i+1}, st^2_{i+1} \ldots st^n_{i+1}$. All other places are 0s. It is easy to initialize these sequences to the initial state of the machine.



fmt	1	0	0	0	0	0	⋯	0	0	1
tp^1	0	1	1	0	1	1	⋯	0	0	0
tp^2	0	1	0	0	0	0	⋯	0	1	0
tp^3	0	0	0	1	0	1	⋯	1	0	0
pos	0	0	0	1	0	0	⋯	0	0	0
st^1	0	0	0	0	0	0	⋯	0	0	0
st^2	0	0	0	1	0	0	⋯	0	0	0

An eight-symbols four-states Turing machine configuration and its encoding

Let us now consider the transition function of the machine. Namely, let *next fmt*, *next* tp^1, *next* tp^2, etc. be the functions mapping the encoding of a configuration – i.e. the tuple of sequences $(fmt, tp^1, tp^2 \ldots)$ – to each of the sequences that constitute the encoding of the configuration after one step of the machine. Observe that, if in configuration $c = (fmt, tp^1, tp^2 \ldots)$ the machine is located at position p, then for all $j \neq p-1, p, p+1$ the transition function does not alter the j-th position of each of the sequences of the encoding, namely $(next\spadesuit(c))_j = \spadesuit_j$.

As a consequence, for all i, the value of $(next\spadesuit(c))_i$ depends only on the values of the sequences fmt, tp^1, tp^2, etc. at places $i-1$, i, and $i+1$. Yet another way to say this is that $(next\spadesuit(c))_i$ depends only on the values of the shifted sequences $\left\{\begin{array}{llll} fmt \gg 1 & tp^1 \gg 1 & tp^2 \gg 1 & \text{etc.} \\ fmt & tp^1 & tp^2 & \text{etc.} \\ fmt \ll 1 & tp^1 \ll 1 & tp^2 \ll 1 & \text{etc.} \end{array}\right\}$ at the place i. Therefore, by Lemma 2, $next\spadesuit(c)$ is a Diophantine function of all these shifted sequences, and, since the shift operation is itself Diophantine, $next\spadesuit$ is Diophantine.

Exercise: the argument in the paragraph above is not strictly correct, because the shifted sequences do not have all the same length (the length of $x \ll 1$ is one less than the length of x, the length of $x \gg 1$ is one more). Therefore we can not apply Lemma 2, which requires sequences of the same length. Fix this problem by padding the sequences with 0s.

In conclusion, for any fixed Turing machine M, we can say that a configuration is encoded by a sequence of Boolean $(m+n+2)$-tuples, namely the sequence $c = c_1, c_2 \ldots$ with

$$c_i = (fmt_i, tp^1_i, tp^2_i \ldots tp^m_i, pos_i, st^1_i, st^2_i \ldots st^n_i)$$

we identify this sequence of tuples with the tuple of sequences

$$(fmt, tp^1, tp^2 \ldots tp^m, pos, st^1, st^2 \ldots st^n)$$

which in turn is just a tuple of $m+n+2$ numbers. We proved that the function

$$\begin{aligned} next\colon \left(\{0,1\}^{m+n+2}\right)^\star \supseteq \text{encodings} &\to \text{encodings} \subseteq \left(\{0,1\}^{m+n+2}\right)^\star \\ c &\mapsto \text{the unique } c' \text{ such that } c \xrightarrow{M} c' \end{aligned}$$

is *local* and Diophantine, where clearly the notation $c \xrightarrow{M} c'$ means "c' is reached by M from c in one step", and *local* means that $(next(c))_i$ depends only on c_{i-1}, c_i, and c_{i+1}.

Two details are very important for our application.

1 - We can assume that the transition function of the machine is such that the machine will never attempt to move past one of the ends of the tape. It is easy to guarantee this condition observing that our transition function has access to the sequence fmt which signals the ends of the tape, and this, in fact, is the reason why fmt was introduced in the first place. To be precise, we define $next$ in such a way that, if the machine would move past the end of the tape, then nothing happens: the current cell remains the same, no symbol is written, the current state does not change.

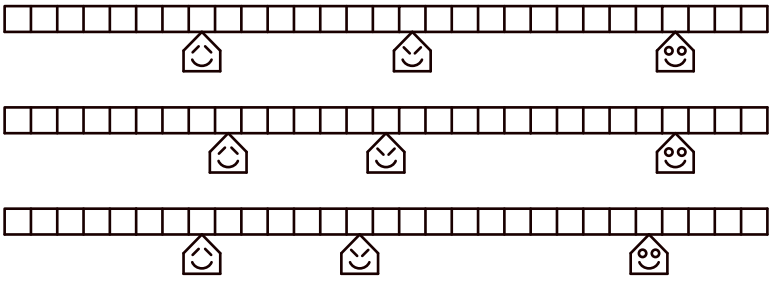
2 - Due to the local nature of our construction, the domain of the function $next$ needs not to be restricted to the set of sequences of tuples encoding valid configurations. Notably, there is no need that the sequence pos contains precisely one 1. Notationally, we will write $\overline{next}$ to mean the function defined locally to match $next$ and having as domain the entire set $\left(\{0,1\}^{m+n+2}\right)^\star$.

5. Iteration by concatenation

Thanks to the encoding of the previous section, given a Turing machine M, we can now say with a Diophantine formula "configuration b is reached from configuration a in *one* step of M", in symbols $a \xrightarrow{M} b$. In this section, we will

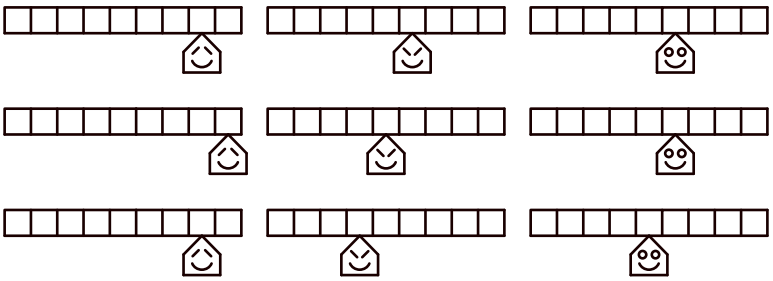
learn how to say "configuration b is reached from configuration a in *any number* of steps of M", denoted $a \xrightarrow{M^*} b$. Doing so is apparently non-trivial, because Diophantine sets are defined by existential formulas, "*there is* a solution of such and such system of equations", however we want to express a property of type $\exists\forall$, namely "*there is* a sequence of configurations beginning with a and ending with b, such that *each configuration in the sequence* is the result of a step of the machine on the previous one".

Our first observation is that, because of the local character of the function $\overline{next}$, if we violate our encoding slightly by placing several instances of the Turing machine on the same tape, and then we compute one step using $\overline{next}$, then all these co-existing instances will make one step in parallel. This, of course, as long as they don't collide! By placing more than one instance, we mean placing more than one 1 on the *pos* sequence in the positions corresponding to the various instances, and encoding their respective states in the corresponding places of the sequences st^i. Indeed, by construction, $(\overline{next}(c))_i$ only depends on c_{i-1}, c_i, and c_{i+1}. Therefore, as long as there is no more than one instance of the machine in each three consecutive cells, the function $\overline{next}$ will never notice that there is more than one instance on the entire tape.



A tape with three machines happily working in parallel (two consecutive steps)

Similarly, we see that our function $\overline{next}$ is ideally suited to compute one step of multiple instances of the machine M operating on multiple separate tapes. In fact, if the tapes have lengths $l_1, l_2 \dots l_n$, it suffices to choose as *fmt* the sequence $1\underbrace{00\cdots0}_{l_1}11\underbrace{00\cdots0}_{l_2}1\cdots1\underbrace{00\cdots0}_{l_n}1$.



Three machines happily working in parallel on separate tapes (two consecutive steps)

Formally, we have seen that, given configurations $c_1, c_2 \dots c_n$

$$\overline{next}(c_1^\frown c_2^\frown \cdots^\frown c_n) = next(c_1)^\frown next(c_2)^\frown \cdots^\frown next(c_n)$$

Finally it is time to express the relation $a \xrightarrow{M^\star} b$. By definition we have

$$a \xrightarrow{M^\star} b \iff \exists\, k\, \exists\, c_1, c_2 \dots c_k \begin{cases} a \xrightarrow{M} c_1 \\ c_1 \xrightarrow{M} c_2 \\ \quad\vdots \\ c_k \xrightarrow{M} b \end{cases}$$

in other words

$$a \xrightarrow{M^\star} b \iff \exists\, k\, \exists\, c_1, c_2 \dots c_k \;\; \overline{next}(a^\frown c_1^\frown \cdots^\frown c_k) = c_1^\frown c_2^\frown \cdots^\frown b \tag{2}$$

this formula is still not satisfactory, because the number of quantified variables is not fixed – $\exists\, c_1, c_2 \dots c_k$ – therefore it is not, prima facie, a Diophantine representation. Nevertheless, the concatenation $c_1^\frown \cdots^\frown c_k$ can be written using a single variable.

$$a \xrightarrow{M^\star} b \iff \exists\, x \;\; \overline{next}(a^\frown x) = x^\frown b \tag{3}$$

and *this* is the Diophantine representation that we have been looking for.

Exercise: the argument in the paragraph above is not strictly correct. In fact, the variables $c_1 \dots c_k$ *existentially quantified in* (2) *are meant to range over all configurations, however the variable* x *in* (3) *ranges over all sequences of* $(m+n+2)$*-tuples. A priori, this might introduce spurious solutions for* x *which do not represent a concatenation of valid configurations. Prove – it is an easy induction – that, as long as* a *and* b *represent configurations and* $\mathrm{len}(a) = \mathrm{len}(b)$*, this is not the case.*

6. Every r.e. set is Diophantine

We are now at the end of our journey. By definition, a set $S \subseteq \mathbb{N}$ is r.e. if there is a Turing machine M such that S is the set of all numbers x with the property that M halts on input x. To be precise, "M halts on input x" means that the machine M will eventually reach a halting state when presented with a half-infinite tape containing, at the beginning, the binary representation of x followed by the special *terminator symbol* $\triangleleft$. Clearly, if this happens, the machine can only explore a finite initial segment of the tape before reaching the halting state, therefore we can equivalently say that S is r.e. if there is a machine M such that $x \in S$ if and only if, for some l, when presented a finite tape of length l containing x encoded as above, the machine M will reach a halting state without attempting to push past the end of the tape.

Now fix a r.e. set S and a machine M deciding S. Remembering that the function mapping x to the sequence containing its binary encoding $(x)_2$ is Diophantine, we have just said that

$$x \in S \iff \exists a, b \begin{cases} \text{the tape of configuration } a \text{ is } (x)_2{}^\frown \triangleleft {}^\frown \overbrace{00\cdots0}^{l} \text{ for some } l \\ M \text{ in } a \text{ is in the initial state and located at position 1} \\ a \xrightarrow{M^\star} b \\ M \text{ is in a halting state in } b \end{cases}$$

The first two conditions are immediately Diophantine, and we have proven that also the third is Diophantine. It is easy to see that also the fourth condition

is Diophantine. The most direct way to this end is to assume, without loss of generality, that the machine M has only one halting state, and this is denoted by number $2^n - 1$. Then the condition becomes $st^1 = st^2 = \cdots = pos$, where $b = (fmt, tp^1 \ldots, pos, st^1 \ldots)$. This establishes the main theorem.

Theorem 3. *All r.e. subsets of* $\mathbb{N}$ *are Diophantine.*

Corollary 4. *Satisfiability of Diophantine equations in undecidable.*

Proof. Exercise. □

References

[Matiyasevich] Yuri V. Matiyasevich *Hilbert's Tenth Problem* The MIT Press, 1993.