ELSEVIER

Theoretical Computer Science 168 (1996) 267–301

Theoretical Computer Science

Small universal register machines

Ivan Korec

Mathematical Institute, Slovak Academy of Sciences, Stefánikova 49, 814 73 Bratislava, Slovakia

Abstract

Several small universal register machines are constructed. The number of instructions vary from 32 to 14, depending on the chosen instruction base and the chosen notion of universality. The proof uses a special coding function for finite sequences of positive integers and some strong classical results concerning distribution of primes.

1. Introduction and the main theorem

The notion of register machines will be discussed in details later, here they are described only very briefly. Every register machine has a finite control unit and uses finitely many registers from the infinite sequence $R_0, R_1, R_2, \ldots$. Each register contains an (arbitrarily large) nonnegative integer. (The set of nonnegative integers will be denoted by $\mathbb{N}$.) Register machines work in discrete time and are deterministic (analogously as, e.g. Turing machines). Various classes of register machines differ in the class of allowed one-step tests and/or operations. In our basic (and most usual) variant of register machines the first three from the following possibilities will be used:

- $\boxed{Ri\,P}$ Add 1 to the content of a register, $R_i := R_i + 1$,
- $\boxed{Ri\,M}$ Subtract 1 from the content of a register, $R_i := R_i \dot{-} 1$. If the original content of R_i is 0 it remains unchanged.
- $\langle Ri \rangle$ Test whether the content of R_i is positive or not. The new inner state depends on the result of the test.
- $\langle \boxed{Ri\,ZM} \rangle$ Test whether the content of R_i is positive or not, and subtract 1 from the content of R_i in the first case. The new inner state depends on the result of the test. (In essential, $\langle \boxed{Ri\,ZM} \rangle$ joins $\langle Ri \rangle$ and $\boxed{Ri\,M}$, and can replace both.)

To every $n \in \mathbb{N}$ and every R-machine M an n-ary partial function Φ_M^n is associated; we shall say that M computes it. To obtain the value $y = \Phi_M^n(x_1, \ldots, x_n)$, the machine starts with $x_1, \ldots, x_n$, in the registers $R_1, \ldots, R_n$, and with zeros in all other registers; the initial inner state is q_1. If the machine M halts in the inner state q_0 then the value y is contained in R_0. Otherwise, e.g. by halting in another inner state, the value y is

PII S0304-3975(96)00080-1

not defined. It is well known that R-machines compute all partial recursive functions (and only them, of course).

Let $(\phi_0, \phi_1, \phi_2, \ldots)$ be a fixed admissible enumeration of the set of unary partial recursive functions (this is a classical notion, but we explain them in the next section). Various approaches to the notion of universal machine differ in the type of allowed coding. Now we shall provide only two variants:

Definition. (i) A register machine M will be called strongly universal if there is a recursive function g such that for all $x, y \in \mathbb{N}$ we have $\phi_x(y) = \Phi_M^2(g(x), y)$.

(ii) A register machine M will be called universal if there are recursive functions f, g, h such that for all $x, y \in \mathbb{N}$ we have $\phi_x(y) = f\left(\Phi_M^2(g(x), h(y))\right)$.

Typical functions f, h for (ii) are $f(x) = \log_2(x)$, $h(x) = 2^x$.

By [11, 13] there are strongly universal R-machines with three registers and universal R-machines with two registers; these numbers of registers are minimal possible. However, we shall not deal with the numbers of registers. The main question which we shall study is

"*How many instructions has the smallest universal R-machine?*"

We do not hope to give a definitive answer but only some upper bounds; the situation is similar as in small universal Turing machines. Further, the answer strongly depends on the forms of instructions and therefore they are explicitly mentioned in the theorem below. (The instruction codes used in (b4) will be explained in the next section.)

Main Theorem. (a) *There are strongly universal register machines*

(a1) *with 32 instructions of the forms* $\boxed{\mathrm{R}i\,\mathrm{P}}$, $\boxed{\mathrm{R}i\,\mathrm{M}}$, $\langle \mathrm{R}i \rangle$;

(a2) *with 22 instructions of the forms* $\boxed{\mathrm{R}i\,\mathrm{P}}$, $\overline{\langle \mathrm{R}i\,\mathrm{ZM} \rangle}$;

(a3) *with 21 instructions of the forms* $\boxed{\mathrm{R}i\,\mathrm{P}}$, $\langle \mathrm{R}i \rangle$, $\overline{\langle \mathrm{R}i\,\mathrm{ZM} \rangle}$.

(b) *There are universal register machines*

(b1) *with 29 instructions of the forms* $\boxed{\mathrm{R}i\,\mathrm{P}}$, $\boxed{\mathrm{R}i\,\mathrm{M}}$, $\langle \mathrm{R}i \rangle$;

(b2) *with 20 instructions of the forms* $\boxed{\mathrm{R}i\,\mathrm{P}}$, $\overline{\langle \mathrm{R}i\,\mathrm{ZM} \rangle}$;

(b3) *with 19 instructions of the forms* $\boxed{\mathrm{R}i\,\mathrm{P}}$, $\langle \mathrm{R}i \rangle$, $\overline{\langle \mathrm{R}i\,\mathrm{ZM} \rangle}$;

(b4) *with 14 instructions of the forms* $\overline{\langle \mathrm{R}i\mathrm{PZ} : \mathrm{R}k \rangle}$.

The machines above can be (and will be) effectively constructed. The machine U_{32} for the statement (a1) is displayed in Fig. 1. Later we shall see that "Stop" need not be counted as an instruction.

The plan of the paper is as follows.

In Section 2 various variants of register machines and of their universality will be discussed. Section 3 contains some necessary number theoretical results, e.g. concerning distribution of primes. Also a special coding function **F** for finite sequences of positive integers is introduced.

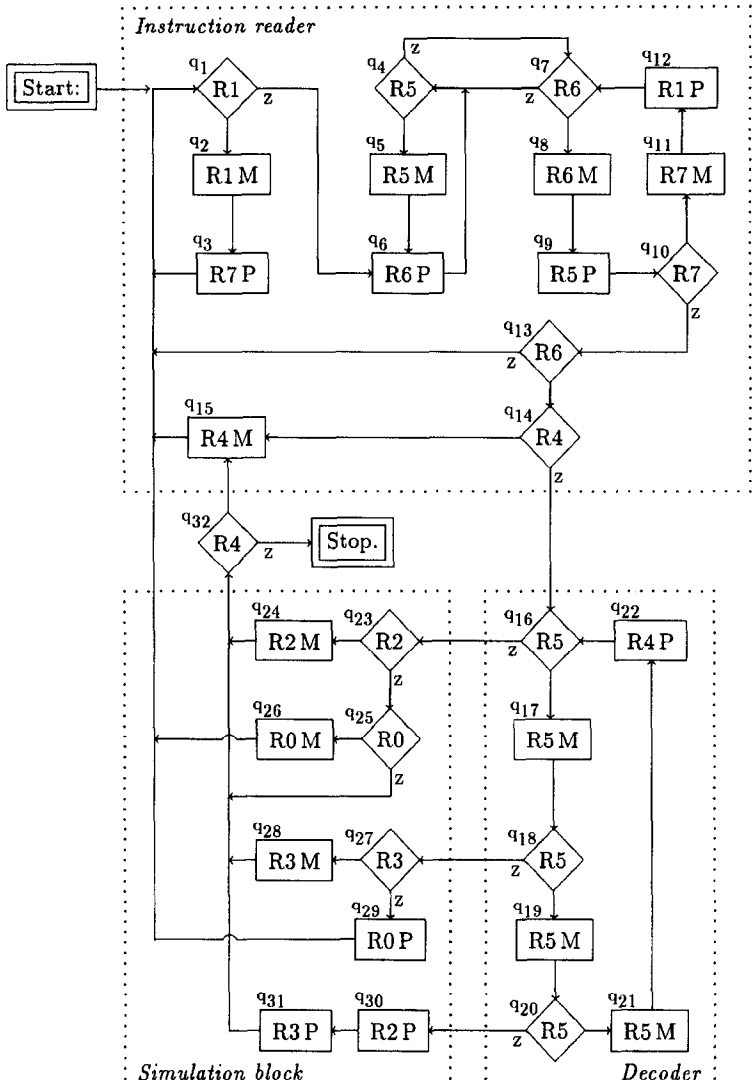

Fig. 1. The universal register machine U_{32}.

The proof of the part (a1) of the Main Theorem will be divided into three following sections.

(1) In Section 4 the so-called R3a-machines are investigated. We prove that every unary partial recursive function is computable by an R3a-machine. We shall use the classical idea from [13] or [11], but the proof will be a little tricky because the instruction base of R3a-machines is rather artificial.

(2) In Section 5 we introduce enumeration of (some) R3a-machines and explain how the machine U_{32} simulates them. The content x of R_1 will be considered as the

number of the simulated R3a-machine M_x. The simulation is correct only for some x; let us call them "good", and let us denote the set of all "good" x by W.

(3) In Section 6 we prove that every R3a-machine M is equivalent with an R3a-machine M_x with the number $x \in W$ (in the sense that they both compute the same unary partial function). A construction of M_x (and of x) will be given. However, the proof that the construction is resultative (after a formalization: a minimalisation used here is regular) needs some strong results about distribution of primes. The proof of (a1) will be finished by application of Chinese Remainder Theorem.

In Section 7 the remaining parts of the Main Theorem (except (b4)) will be proved. The necessary machines are easily obtained from U_{32}. Also a strongly universal register program (defined below) with 42 instructions is presented; it is obtained by a transformation of U_{32}. The part (b4) is considered in Section 8. The last two sections contain some remarks about lower bounds, other variants, etc.

2. Variants of register machines and of their universality

The aim of the present section is to give an overview of variants of register machines which has been considered in the literature (preferably in textbooks) or will be important for the present paper. A similar overview will be given about the notion of universal register machines. Of course, the presented overview will not be complete. No *formal theory* of classes of register machines will be build, hence general definitions here need not be given very strictly. However, at least those concrete classes of R-machines which will appear in the theorems must be defined precisely.

Register machines (R-machines) use registers from the infinite sequence of registers $R_0, R_1, R_2, \ldots$. Every R-machine really uses only finitely many of them but we may imagine that it has as many registers as it need. Each register contains an (arbitrarily large) nonnegative integer (but we may assume that at each moment only finitely many registers contain nonzero integer). We shall say that a register is empty if it contains zero. We shall usually use the same symbol for a register and its content. Besides these registers a register machine has a finite control unit. R-machines work in discrete time and are deterministic. In every step an R-machine can tests the contents of registers, and can change these contents and the state of its control unit (called also "inner state"). The inner states will be chosen from the infinite sequence $q_0, q_1, q_2, q_3, \ldots$; this restriction is not substantial but it is technically suitable. From similar reason we shall fix the start state q_1 and the final state q_0.

The strength of a class of R-machines depends on the chosen instruction base. If we allow very strong instructions then the obtained class would not correspond to the (informal) idea of register machines. From the obvious reasons we do not consider any non-recursive operations. However, even some primitive recursive operations could enable such "R-machines" to simulate, e.g., many-tape many-dimensional Turing machines in real time (and we do not wish that). It also seems reasonable not to allow any strong arithmetical operations (to avoid random access machines) as well as any

second-order addressing (to avoid RASP machines, "random access stored program"). We allow only very elementary operations and tests, each with a small number of registers. (The bound, e.g. at most 2 or 3 registers in one instruction, concern the class of machines, and not only a particular machine. For the classical R-machines it is equal to 1.) We shall usually speak simply about operation codes instead "operation and test codes". We can see that some operation codes contain parameters which must be replaced by numbers of registers (they can be considered as "schemes of codes"); some other do not contain such parameters, and are used always with the same registers. Although the operation codes are composed of several letters or digits (to remember their semantics more easily) they ought to be understood as unique symbols.

Various classes of R-machines differ in the class of allowed one-step operations. Four operation codes $\boxed{Ri\,P}$, $\boxed{Ri\,M}$, $\langle Ri \rangle$, $\langle Ri\,ZM \rangle$ were introduced in the previous section; we shall also use the following ones:

$\boxed{Ri := Rk}$ Assign the content of R_k to R_i; the content of R_k is not changed.

$\boxed{Ri := 0}$ Assign 0 to R_i.

$\langle Ri = Rk \rangle$ Test whether the contents of the registers R_i, R_k are equal. The new inner state depends on the result of the test (it is q_{k_2} by equality).

$\langle RiPZ : Rk \rangle$ Test whether the contents of the registers R_i, R_k are equal. If yes, put 0 into R_i, otherwise add 1 to R_i (the content of R_k is not changed). The new inner state depends on the result of the test; it is q_{k_2} if the original contents of R_i, R_k were equal, q_{k_1} otherwise.

$\langle R20ZM \rangle$ The instruction tests whether $R_2 = 0$ and if yes, it tests whether $R_0 = 0$. If a positive content is found, 1 is subtracted from it. The next inner state will be q_{k_1} if R_2 was positive or if both R_2, R_0 contained zero. Otherwise the next state will be q_{k_2}.

$\langle R3ZM0P \rangle$ The instruction tests whether $R_3 = 0$ and if yes adds 1 to R_0; the next inner state will be q_{k_2}. Otherwise 1 is subtracted from R_3 and the next state is q_{k_1}.

$\boxed{R23P}$ The instruction adds 1 to both R_2 and R_3; the next inner state will be q_j.

$\boxed{R023P}$ The instruction adds 1 to all three registers R_0, R_2 and R_3.

$\langle R0P2ZM \rangle$ The instruction adds 1 to R_0 and then performs $\langle R2\,ZM \rangle$.

$\langle R0M3ZM \rangle$ The instruction subtracts 1 from R_0 (if possible) and then performs. $\langle R3\,ZM \rangle$.

$\langle R23P0Z \rangle$ The instruction adds 1 to both R_2, R_3 and then tests whether R_0 is empty (like $\langle R0 \rangle$).

$\langle R023\,ZM \rangle$ If the contents of all three registers R_0, R_2, R_3 are positive, 1 is subtracted from every of them, end the next inner state will be q_{k_1}. Otherwise the contents remain unchanged and the next inner state is q_{k_2}.

$\langle R02ZM \rangle$ Like above, but only with the registers R_0, R_2.

Most of the operation codes above are collected from some textbooks, e.g. [11, 13]. We have used rectangles $\boxed{}$ for purely operational codes of instructions with $n = 1$

in definition (2.1) and hexagons ⟨ ⟩ (or rhomboids ◇) for instructions (2.1) with $n \geqslant 2$, i. e. containing tests. (The bigger forms will be used in flowcharts, the smaller one in the text and formulas.) In flowcharts the arrows corresponding to q_{k_2} will be denoted by 'z' (from "zero").

Remark. Rather artificial codes ⟨R20ZM⟩, ⟨R3ZM0P⟩, [R023P] corresponds to the activity of the simulation block of U_{32}, see Fig. 1. The role of other artificial codes is similar.

Definition 2.1. (i) Instructions of register machines will be ordered $(n+2)$tuples of the forms

$$(q_j, X, q_{k_1}, \ldots, q_{k_n}), \tag{2.1}$$

where X is an operation code, n is determined by X (usually $n \leqslant 2$) and $j, k_1, \ldots, k_n$ are nonnegative integers.

(ii) A register machine is a finite set of instructions which does not contain two distinct elements with the same first component.

(iii) A class of register machines can be specified by the set of allowed operation codes and by the conditions on $j, k_1, \ldots, k_n$ (which may be formulated for the whole machines).

We can consider the first component of every instruction (2.1) as the address where the other components (the operation code and the next addressee(s)) can be found. The condition in (ii) expresses that register machines are deterministic. The parts (i) and (ii) can be considered as real definition (provided we ask that X is taken from the list above or another similar fixed list). However, the part (iii) only shows an idea, and cannot be a base for a theory. Some conditions can be formulated for single instructions. However, some other (e.g. that concerning the forms of allowed flowcharts, in particular: allowed jumps) must consider whole machines.

If a class of machines is specified we can speak about their *computations* as finite or infinite sequences of *configurations*. We shall omit these trivial but tedious considerations. Further we can determine *the computed (partial) functions* Φ_M^n for M from the considered class and suitable n (usually at least for $n = 1$, very often for all $n \in \mathbb{N}$). We shall do that like in the previous section; any change will be explicitly mentioned. The computations which are used to compute values of Φ_M^n called *normal computations* (the considered values n must be clear from the context). In particular, they usually start in the initial state q_1.

2.2. *Variants of register machines*

2.2.1. Classical register machines use all instructions (2.1) for

$$X \in \{[\mathrm{R}i\,\mathrm{P}], [\mathrm{R}i\,\mathrm{M}], \langle \mathrm{R}i \rangle\}.$$

There are no special restrictions on i, j, k_1, k_2.

2.2.2. Modified register machines are defined similarly, only we use

$$X \in \{\boxed{\mathrm{R}i\,\mathrm{P}}, \langle \mathrm{R}i\,\mathrm{ZM} \rangle\}.$$

2.2.3. The class of machines used in (a3) and (b3) of the Main Theorem is specified by

$$X \in \left\{\boxed{\mathrm{R}i\,\mathrm{P}}, \diamondsuit_{\mathrm{R}i}, \langle \mathrm{R}i\,\mathrm{ZM} \rangle\right\}.$$

It is not so nice that the two classes above because the set of operation codes is superfluous: it can be reduced without any change of the strength of the class. We do not introduce a special name for them.

2.2.4. A class of register machines well corresponding to the *unlimited register machines* of [19] uses

$$X \in \{\boxed{\mathrm{R}i := 0}, \boxed{\mathrm{R}i\,\mathrm{P}}, \boxed{\mathrm{R}i := \mathrm{R}k}, \langle \mathrm{R}i = \mathrm{R}k \rangle\}$$

(the corresponding codes used there are $Z(i)$, $S(i)$, $T(i,k)$, $J(i,k,k_2)$). The instructions are linearly ordered (so that they are enumerated by positive integers $1,2,3,\ldots$) and are executed in this order unless a conditional jump is performed. This fact can be expressed by the condition $k_1 = j+1$ in (2.1). Like above, there is no stop instruction; the machine halts when it ought to perform a non-existing instruction. Notice that in this case we could rather speak about *programs* than about *machines*.

2.2.5. The same as in 2.2.4 can be done also with the instruction codes from 2.2.1. We shall use the term *register programs* for the obtained machines. (In 2.2.4 we preserved the term used in the literature.)

2.2.6. A class of register machines well corresponding to the *bar bones programming language* of [1] uses

$$X \in \left\{\boxed{\mathrm{R}i\,\mathrm{P}}, \boxed{\mathrm{R}i\,\mathrm{M}}, \diamondsuit_{\mathrm{R}i}\right\};$$

the book uses incr i and decr i for the first two instructions. For the control WHILE statements with conditions $\mathrm{R}_i = 0$ are used; otherwise the instructions are performed in the given order. We can simulate that by the condition $k_1 = j+1$ (which will hold with some exceptions). WHILE statements can also be simulated, but we must ask that the jumps are nested in suitable way (some conditions on the allowed form of the flowcharts can be formulated).

2.2.7. A very similar class of register machines can be associated to the *abacus machines* of [2]. The book uses symbols a_i and s_i for the first two instructions, and $(\cdots)_k$ for the WHILE statement with the condition $R_k = 0$.

2.2.8. R3-machines use three registers R_0, R_2, R_3 and

$$X \in \left\{\langle \mathrm{R0\,ZM} \rangle, \langle \mathrm{R2\,ZM} \rangle, \langle \mathrm{R3\,ZM} \rangle, \boxed{\mathrm{R023P}}\right\}.$$

There are also some specific conditions on the subscripts in (2.1); the most important is $k_2 = k_1 + 1$ whenever $n = 2$, the other, e.g., arrange that an R3-machine never halts

in a nonfinal state. R3-machines compute all unary partial recursive functions (they use R_2 as the input register). They roughly correspond to the operator algorithms with the operations $:2$, $:3$, $:5$ and $\times 30$. (They would correspond better if the condition $k_2 = k_1 + 1$ is omitted.) R3-machines were used in [4] to construct a (strongly) universal register machine with 37 instructions. (The register R_1 was deleted on purpose; the universal machine uses it for the number of the simulated R3-machine.)

2.2.9. R3a-machines are similar to R3-machines above, and they also will be used similarly. They use the same three registers but use operation codes

$$X \in \left\{ \langle \overline{\text{R20ZM}} \rangle , \langle \overline{\text{R3ZM0P}} \rangle , \boxed{\text{R23P}} \right\} .$$

Specific restrictions (also similar to that of R3-machines) will be discussed in Section 4. This rather artificial class is only introduced to prove the Main Theorem. (In the name "R3a-machines" the digit 3 expresses the number of used registers and "a" is used to distinguish the class from the previous one; below we shall continue similarly.)

2.2.10. R3b-machines are very similar to R3a-machines; they use

$$X \in \left\{ \langle \overline{\text{R0P2ZM}} \rangle , \langle \overline{\text{R0M3ZM}} \rangle , \langle \overline{\text{R23P0Z}} \rangle \right\} .$$

They are in some sense nicer than R3a-machines, and they could be used to prove (a1) and (b1) of Main Theorem. However, they are not so suitable for the proof of the other parts. The corresponding machine U'_{32} can be obtained from U_{32} if we replace the original simulation block by that given in the left-hand part of Fig. 8.

2.2.11. R3c-machines are similar to R3-machines but the used operation codes are

$$X \in \left\{ \boxed{\text{R0 P}} , \boxed{\text{R2 P}} , \boxed{\text{R3 P}} , \langle \overline{\text{R023 ZM}} \rangle \right\} .$$

They roughly correspond to the operator algorithms with the operations $\times 2$, $\times 3$, $\times 5$ and $:30$.

2.2.12. R2-machines use the registers R_0, R_2 and

$$X \in \left\{ \langle \overline{\text{R0 ZM}} \rangle , \langle \overline{\text{R2 ZM}} \rangle , \boxed{\text{R02 P}} \right\} .$$

They compute all unary partial recursive functions on the set $\{2^x \mid x \in \mathbb{N}\}$ of the powers of 2. Hence they compute all unary partial recursive functions provided the input or output x is coded as 2^x. They are used in the proofs of the parts (b1)–(b3) of the Main Theorem (and were used similarly in [4]). Like R3-machines, they use R_2 as the input register. They correspond to the operator algorithms with the operations $:2$, $:3$, and $\times 6$.

2.2.13. R2a-machines use the registers R_0, R_2 and

$$X \in \left\{ \boxed{\text{R0 P}} , \boxed{\text{R2 P}} , \langle \overline{\text{R02ZM}} \rangle \right\} .$$

They compute all unary partial recursive functions on the set $\{2^x \mid x \in \mathbb{N}\}$ of the powers of 2. Hence they compute all unary partial recursive functions provided the

input or output x is coded as 2^x. Like above, they use R_2 as the input register. They correspond to the operator algorithms with the operations $\times 2$, $\times 3$, and $:6$.

2.2.14. R2b-machines use the registers R_0, R_2 and

$$X \in \{\langle \overline{R0PZ : R2} \rangle, \langle \overline{R2PZ : R0} \rangle\}.$$

They must use the same register for input and output; let it be the register R_0. Also the condition $k_2 = k_1 + 1$ is required. R2b-machines *do not compute* (without coding) all unary partial recursive functions on any infinite (recursive) subset of $\mathbb{N}$ because $\max(R_0, R_2)$ cannot decrease during any computation. However, they compute all unary partial recursive functions if we use "the exponent of 2 in the factorization of x" for the output decoding. They are used in the proof of the part (b4) of the Main Theorem.

The classes of machines 2.2.1–2.2.7 are in some sense natural. Some of the further classes of machines are rather artificial; they may be useful in some proofs but they are not suitable, e.g., as a basic model for teaching.

At some places above *operator algorithms* were mentioned. They are considered e.g. in [11]. An operator algorithm works with one non-negative integer variable x and it iteratively applies given operations and tests on it (so it could be considered as a one-register machine, but the operations are rather strong). The operation $:n$ means "if $n|x$ then divide x by n, otherwise do not change it", and the next inner state depends on the test; the meaning of $\times n$ is clear. The idea of operator algorithms with the mentioned instructions is standardly used in simulation of many-register machines by two-register ones (we shall also use it).

Now we shall consider various variants of the notion of universal register machines. Let $(\phi_0, \phi_1, \phi_2, \ldots)$ be a fixed admissible enumeration of the set of unary partial recursive functions. (For example, ϕ_i can be the unary partial recursive function computed by the ith Turing machines; we can consider any usual variant of Turing machines, any usual I/O codings and any usual enumeration of chosen Turing machines. We can also effectively enumerate the classical R-machines defined above, and define ϕ_i as the unary partial function computed by the ith R-machine.) Various approaches to the notions of universal partial function and universal machine differ in the type of coding. The difference is often unimportant when a general theory is developed, but may be very substantial when the number of instructions is studied. For R-machines we shall provide several variants (for the completeness we repeat also the variants from Section 1).

Definition 2.3. (i) A register machine M will be called strongly universal if there is a recursive function g such that for all $x, y \in \mathbb{N}$ we have $\phi_x(y) = \Phi_M^2(g(x), y)$.

(ii) A register machine M will be called 2-universal if there are recursive functions h, g such that for all $x, y \in \mathbb{N}$ we have $\phi_x(y) = \Phi_M^2(g(x), h(y))$.

(iii) A register machine M will be called 3-universal if there is a recursive function g_2 such that for all $x, y \in \mathbb{N}$ we have $\phi_x(y) = \Phi_M^2(g_2(x, y))$.

(iv) A register machine M will be called 4-universal if there are recursive functions f, g such that for all $x, y \in \mathbb{N}$ we have $\phi_x(y) = f(\Phi_M^2(g(x), y))$.

(v) A register machine M will be called universal if there are recursive functions f, g, h such that for all $x, y \in \mathbb{N}$ we have $\phi_x(y) = f\left(\Phi_M^2(g(x), h(y))\right)$.

(vi) A register machine M will be called 6-universal if there are recursive functions f, g_2 such that for all $x, y \in \mathbb{N}$ we have $\phi_x(y) = f\left(\Phi_M^1(g_2(x, y))\right)$.

(vii) A register machine M will be called 7-universal if there are $k \in \mathbb{N}$, a unary recursive function f and binary recursive functions $g_1, \ldots, g_k$ such that for all $x, y \in \mathbb{N}$ we have

$$\phi_x(y) = f\left(\Phi_M^k(g_1(x, y), \ldots, g_k(x, y))\right).$$

(viii) A register machine M will be called 8-universal if

$$\left\{\lambda y.\Phi_M^2(x, y) \mid x \in \mathbb{N}\right\} = \left\{\phi_x \mid x \in \mathbb{N}\right\}.$$

Of course, we shall not deal with all notions above; we only wanted to show a wide spectrum of possibilities for the notion of universality. Obviously there are also well-acceptable weaker notions which also suffice, e.g., for undecidability of halting problem. The machines satisfying (iii), (iv), (vi), (vii) or (viii) will be also called *weakly universal* (for (ii) it is better to say "universal"). In (viii) we use λ notation; the left-hand side consists of all unary partial functions obtained from $\Phi_M(x, y)$ when the first argument is fixed. The function h is input coding function and f is output decoding function. The definition above allows f, h to be general recursive (but total). Sometimes they are asked to be primitive recursive but such restrictions seems to be unnecessary; on the other hand, in concrete cases only primitive recursive functions are usually used. We may not allow f, h to be arbitrary partial recursive; in such case the machine M need not compute anything nontrivial. Concerning the functions g, g_i, we cannot reasonably restrict them to primitive recursive functions because the admissible sequence is defined only up to recursive isomorphism. (We shall not consider the possibility of special choice of $(\phi_0, \phi_1, \phi_2, \ldots)$.)

We can summarize this section as follows. The answer to our main question (on the minimal number of instructions of universal register machine) depends on:

(1) The allowed *operation codes*. Here we also determine the used registers. We may allow different possibilities for different registers.

(2) The chosen *flow control*. There are following possibilities:

- Arbitrary flowcharts allowed. (This approach is usual in theoretical computer science. It was used for Turing machines, finite automata, pushdown automata, etc., hence we can consider it as the most natural also for register machines.)
- The instructions are linearly ordered, and are executed in this order until a jump instruction (maybe, a conditional one) is reached. We can speak about programs (it is only a terminological question).
- Structured programs of various kinds, e.g. WHILE cycles allowed.
- Other restrictions. For example, the instructions are linearly ordered, and conditional instructions must always choose one of two consecutive instructions; R3a-machines will belong there.

(3) The chosen *notion of computed functions*; it mostly depends on the input and output coding. Since we consider number functions we can put arguments directly to some registers, and leave the other registers empty. We shall say in this case that no input coding is used. Analogously, the result can directly be found in a register; we shall say that no output code is used. However, we also can use a (nontrivial) input or output coding or both; so we have four possibilities. Usually, the concrete subscripts of input/output registers are not important, but it may be important whether the output register coincides with one of the input registers or not. (However, the equivalence of registers can be destroyed in (1).) In any case, we shall fix I/O coding (including I/O registers) for every considered class of machines.

(4) The chosen *notion of universality of machines*, which is usually related to the notion of a universal partial recursive function. (However, the relationship need not be so direct.) The concrete subscripts of input/output registers are not important (provided that the equivalence of register was preserved in (1)). However, it may be important whether the output register coincides with an input register or not.

By Church thesis we have a big freedom in the choice of above when we want to study (partial) recursive functions. However, very small details can change the numerical answer to the question above.

3. Number theoretical preliminaries.

$\mathbb{N}$ will denote the set of non-negative integers. We shall use the symbols DIV and MOD for the quotient and the rest by integer division. The integer part of a real x will be denoted $\lfloor x \rfloor$. The number of primes not exceeding x will be denoted $\pi(x)$; it is well known (see e.g. [15]) that $\pi(x)$ is asymptotically equal to $x/(\ln x)$; some less known results are contained in Theorem 3.3. We shall also need the following functions (not so commonly used).

Definition 3.1. For all $x, y \in \mathbb{N}$ we denote

$$(3.1.1)\quad \mathrm{MaxPr}(x) = \begin{cases} \text{the maximal prime divisor of } x, \text{if } x > 1, \\ 0 \quad \text{otherwise;} \end{cases}$$

$$(3.1.2)\quad \mathbf{nd}(x, y) = \begin{cases} \text{the } y\text{th non-divisor of } x, & \text{if } x \neq 0, y \neq 0, \\ 0 \quad \text{otherwise;} \end{cases}$$

$$(3.1.3)\quad \mathbf{F}(x, y) = (x + 1) \text{ MOD } \mathbf{nd}(x + 1, y).$$

To illustrate the definition, let us give several examples. We have $\mathrm{MaxPr}(30) = 5$, $\mathrm{MaxPr}(17) = 17$. Now let us compute $\mathbf{F}(29,4)$. Since,

$$\mathbf{nd}(30,1) = 4, \quad \mathbf{nd}(30,2) = 7, \quad \mathbf{nd}(30,3) = 8, \quad \mathbf{nd}(30,4) = 9, \ldots$$

we have $\mathbf{F}(29,4) = 30 \text{ MOD } \mathbf{nd}(30,4) = 30 \text{ MOD } 9 = 3$.

The function MaxPr, **nd**, **F** obviously are primitive recursive (cases for small arguments were written only to arrange that). The function **F** will be used as a coding function, similarly as the Gödel's coding function

$$\Gamma(x,y) = \mathbf{l}(x) \bmod (1 + (y+1)\cdot \mathbf{r}(x))$$

(where **l**, **r** are suitable pairing functions) is usually used. The only advantage of **F** is that we can compute it with a smaller number of instructions.

Lemma 3.2. *For every finite sequence* $(a_1, a_2, \ldots, a_k)$ *of positive integers there is* $x \in \mathbb{N}$ *such that*

$$\mathbf{F}(x,i) = a_i \quad \textit{for all} \quad i = 1,2,\ldots,k.$$

Proof. Let us take k primes $p_1 < p_2 < \ldots < p_k$ such that $p_k < 2p_1$ and $p_i > a_i$ for all $i = 1,\ldots,k$. The number x can be obtained by solving the following systems of congruences:

$$x + 1 \equiv a_i \pmod{p_i} \qquad \text{for } i = 1,\ldots,k;$$

$$x + 1 \equiv 0 \pmod{p^e}, \text{ where } e = \left\lfloor \frac{\ln p_k}{\ln p} \right\rfloor \text{ for all other primes } p < p_k.$$

By the second part (and the inequality $p_1 < p_2 < \ldots < p_k$) the only non-divisors of $x+1$ not exceeding p_k are $p_1,\ldots,p_k$. By the first part they all are non-divisors of $x+1$ and, therefore, we have $\mathbf{nd}(x+1,i) = p_i$ for all $1 \leqslant i \leqslant k$. Then the first part arranges $\mathbf{F}(x,i) = (x+1) \bmod p_i = a_i$. The moduli of the presented system of congruences are pairwise relatively prime, hence the system is solvable by Chinese Remainder Theorem. □

We shall need the following results about distribution of primes; they are only easy reformulations of deep number theoretical theorems.

Theorem 3.3. *For every* $\varepsilon > 0$, $\alpha \geqslant 0.55$, $1 > \gamma > \frac{13}{23}$ *and* $1 > \delta > \frac{38}{61}$ *we have*

$$\limsup_{N\to\infty} (\pi(N + N^{\varepsilon}) - \pi(N)) = \infty;$$

$$\lim_{N\to\infty} (\pi(N + N^{\alpha}) - \pi(N)) = \infty;$$

$$\liminf_{N\to\infty} \frac{\pi(N + N^{\gamma}) - \pi(N)}{\frac{N^{\gamma}}{\ln N}} \geqslant \frac{1}{177};$$

$$\lim_{N\to\infty} \frac{\pi(N + N^{\delta}) - \pi(N)}{\frac{N^{\delta}}{\ln N}} = 1.$$

Proof. The first result can be derived from the asymptotic formula for $\pi(n)$. (We can also replace N^{ε} by, e.g., $(\ln N)^{1+\varepsilon}$.) To prove the second result we use the statement

from [15, p. 160] that for every $r \geqslant 0.548$ and all sufficiently large N there is a prime between N and $N + N^r$. The third result can be found in [5], and the fourth one in [3, p. 196 of Russian translation]. □

Theorem 3.4. *Let us denote*

$$A(N,\beta) = \{x \in \mathbb{N} \mid 0 < x \leqslant N \wedge \mathrm{MaxPr}(x) > N^{\beta}\}.$$

Then for every real β, $\frac{1}{2} \leqslant \beta \leqslant 1$ *we have*

$$\lim_{N\to\infty} \frac{\mathrm{card}\, A(N,\beta)}{N} = \ln \frac{1}{\beta}.$$

Proof. Since $\beta \geqslant \frac{1}{2}$ every $x \in A(N,\beta)$ has exactly one prime divisor greater than N^{β}. Therefore, the set $A(N,\beta)$ can be divided into pairwise disjoint subsets of multiples of these primes. The number of multiples of a prime $p \geqslant N^{\beta}$ in $A(N,\beta)$ is equal to $\lfloor \frac{N}{p} \rfloor$ and, hence,

$$\mathrm{card}\, A(N,\beta) = \sum_{N^{\beta} < p \leqslant N} \left\lfloor \frac{N}{p} \right\rfloor.$$

(The summation subscript p here and below runs over primes.) To estimate the sum we shall use the formula

$$\sum_{p \leqslant x} \frac{1}{p} = \ln\ln x + B + \mathrm{O}\left(\frac{1}{\ln x}\right),$$

where B is a constant; see [15, p. 106]. Remember that $\mathrm{O}(f(x))$ denotes a summand with absolute value not exceeding $C \cdot f(x)$ for a constant C. Using this notation, we have

$$\begin{aligned}\sum_{N^{\beta} < p \leqslant N} \left\lfloor \frac{N}{p} \right\rfloor &= \sum_{N^{\beta} < p \leqslant N} \frac{N}{p} + \mathrm{O}\Big(\frac{N}{\ln N}\Big) = N \cdot \sum_{N^{\beta} < p \leqslant N} \frac{1}{p} + \mathrm{O}\Big(\frac{N}{\ln N}\Big) \\ &= N \cdot \left(\ln\ln N - \ln\ln N^{\beta} + \mathrm{O}\left(\frac{1}{\ln N}\right)\right) + \mathrm{O}\Big(\frac{N}{\ln N}\Big) \\ &= N \cdot \ln \frac{\ln N}{\ln N^{\beta}} + \mathrm{O}\Big(\frac{N}{\ln N}\Big) = N \cdot \ln \frac{1}{\beta} + \mathrm{O}\Big(\frac{N}{\ln N}\Big).\end{aligned}$$

This immediately gives the statement of the theorem. □

4. R3A-machines

These machines are rather artificial and the only purpose to introduce them is their role in our construction of the universal R-machine U_{32} (and some derived ones). Their inner states (again) will be $q_0, q_1, q_2, \ldots$, the start state q_1 and the only final state will be q_0. As we shall see, the fixed sequence of inner states is now much more substantial

than it was for the basic variant of R-machines. R3a-machines will use three registers, R_0, R_2 and R_3. The register R_1 is omitted on purpose because the R-machine U_{32} will use it for the number of a simulated machine. (Its content will not be deleted when writing R3a-configurations, but it remains constant and plays no role in the tests during a computation.)

Definition 4.1. An R3a-machine is a finite set of ordered triples resp. quadruples of the forms

$$\left(q_j,\ \langle \underline{\mathrm{R20ZM}} \rangle, q_k, q_{k+1}\right), \qquad \left(q_j,\ \langle \underline{\mathrm{R3ZM0P}} \rangle, q_k, q_{k+1}\right), \qquad \left(q_j, \boxed{\mathrm{R23P}}, q_k\right),$$

$j, k \in \mathbb{N}$, which does not contain two distinct elements with equal first components and which satisfies the conditions:

(1) q_0 is not the first component of any instruction;

(2) if q_j, $j \neq 0$ is the third or the fourth component of any instruction then q_j is also the first components of some instruction;

(3) no instruction has the form $(q_j, \langle \underline{\mathrm{R20ZM}} \rangle, q_0, q_1)$.

The triples or quadruples above will be also called R3a-instructions, or simply instructions. Conditions (1)–(3) are only technical and do not influence the computational strength of R3a-machines. They will be useful by the enumeration of R3a-machines.

The definition of computed unary functions must be modified so that R_2 (and not R_1) is used as the input register. If an R3a-machine stops it will always give a result (in the register R_0). Hence, a non-defined value always requires an infinite computation.

Theorem 4.2. *Every unary partial recursive function is computable by an R3a-machine.*

Proof. We prove a more effective statement than that formulated in the theorem. Namely, we shall show how to construct an R3a-machine which computes a given partial function f (where f is given by its index or, e.g., by a Turing machine which computes f). We shall use the standard techniques (described e.g. in [11] or [13]). The computation of any partial recursive function $f(x)$ will consist of three stages:

(1) Computing 2^x from x.

(2) Computing $2^{f(x)}$ from 2^x; this part will not finish if $f(x)$ is not defined.

(3) Computing $f(x)$ from $2^{f(x)}$.

The stage (1) will be performed by the flowchart in Fig. 2. (The only change will be that the non-zero subscripts of q_i will be enlarged by an additive constant; the constant depends on the computed function.) This part transforms

$$(q_1;\ 0, z, x, 0) \quad \text{into} \quad (q_9;\ 0, z, 0, 2^x).$$

Table 1 contains an example of computation for $x = 2$ where all typical cycles can be found. If an arrow is directed into a bullet • then it will be never used in any reasonable computation, e.g. any one from the initial state. (However, by the definition

of R3a-machines these arrows are directed to some instructions, which can easily be found be the labels q_j. The bullets only help us to understand the activity of a machine.) Notice that the content of R_2 is transferred into R_0 in the first part of the computation. Further, notice that the instruction q_{14} indeed tests only whether $R_0 = 0$ (because $R_2 = 0$ can be predicted). Hence, the arrow labelled by "z" corresponds to *non-zero* content of R_0 in this case.

Let us consider the stage 2 now. The left part of Fig. 3 contains several "macro-instructions" for R3a-machines and its right part contains translations of these macro-instructions into the original R3a-instructions (or previously defined macroinstructions).

Table 1
The computation of the block from Fig. 2 for $x = 2$

0: (q1; 0, 9, 2, 0)	17: (q15; 2, 9, 2, 1)	34: (q12; 1, 9, 4, 0)
1: (q2; 1, 9, 2, 0)	18: (q12; 2, 9, 1, 1)	35: (q13; 1, 9, 3, 0)
2: (q3; 1, 9, 1, 0)	19: (q13; 2, 9, 0, 1)	36: (q15; 1, 9, 4, 1)
3: (q2; 2, 9, 1, 0)	20: (q15; 2, 9, 1, 2)	37: (q12; 1, 9, 3, 1)
4: (q3; 2, 9, 0, 0)	21: (q12; 2, 9, 0, 2)	38: (q13; 1, 9, 2, 1)
5: (q2; 3, 9, 0, 0)	22: (q14; 1, 9, 0, 2)	39: (q15; 1, 9, 3, 2)
6: (q4; 2, 9, 0, 0)	23: (q10; 0, 9, 0, 2)	40: (q12; 1, 9, 2, 2)
7: (q5; 2, 9, 1, 1)	24: (q11; 0, 9, 0, 1)	41: (q13; 1, 9, 1, 2)
8: (q14; 2, 9, 0, 1)	25: (q6; 0, 9, 1, 2)	42: (q15; 1, 9, 2, 3)
9: (q10; 1, 9, 0, 1)	26: (q7; 0, 9, 1, 1)	43: (q12; 1, 9, 1, 3)
10: (q11; 1, 9, 0, 0)	27: (q8; 0, 9, 2, 2)	44: (q13; 1, 9, 0, 3)
11: (q6; 1, 9, 1, 1)	28: (q10; 0, 9, 2, 1)	45: (q15; 1, 9, 1, 4)
12: (q7; 1, 9, 1, 0)	29: (q11; 0, 9, 2, 0)	46: (q12; 1, 9, 0, 4)
13: (q8; 1, 9, 2, 1)	30: (q6; 0, 9, 3, 1)	47: (q14; 0, 9, 0, 4)
14: (q10; 1, 9, 2, 0)	31: (q7; 0, 9, 3, 0)	48: (q9; 0, 9, 0, 4)
15: (q12; 2, 9, 2, 0)	32: (q8; 0, 9, 4, 1)	
16: (q13; 2, 9, 1, 0)	33: (q10; 0, 9, 4, 0)	

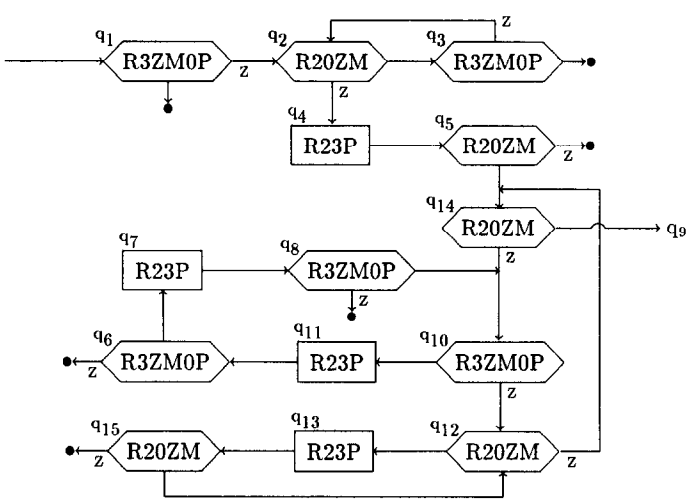

Fig. 2. The block of R3a-machines for 2^x.

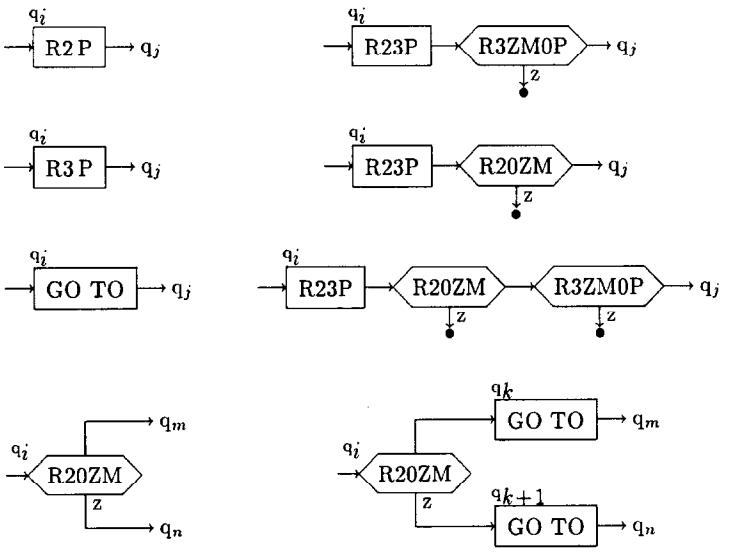

Fig. 3. Macroinstructions in R3a-machines.

The bullets • are used as above. To understand the fourth macroinstruction, remember that the left part is an R3a-instruction only if $n = m+1$. Otherwise we have a branching not allowed by the definition of R3a-machines. However, we can replace such general branching by seven original instructions (every GOTO replaces three instructions, as is shown above). Of course, we shall generalize the instruction $\langle$R3ZM0P$\rangle$ similarly. Then we can use inner states as freely as we could in the original R-machines.

The first two "macroinstructions" are the instructions [R*i* P] for $i = 2$ and $i = 3$. If we could similarly translate all three instructions [R*i* P], [R*i* M], $\diamond$R*i*$\diamond$ for all $i = 0, 2, 3$ then we could use this fact to prove that R3a-machines compute all (unary) partial recursive functions. However, such translations do not exist in general (at least such simple ones). Therefore, we have to prove the theorem in a modified way.

In the second stage we can use only two counters R_2, R_3 in essential, but we can imagine 2^x as e.g. $2^x \cdot 3^0 \cdot 5^0 \cdot \ldots \cdot p_n^0$, and the exponents can be considered as contents of n registers. To work with them, we must be able to multiply and to divide by some constants; the necessary (schemes of) blocks are given in Fig. 4. The upper block performs multiplication, more precisely, it transforms

$$(q_i;\ y,z,0,x) \quad \text{into} \quad (q_j;\ y+1,z,ax+b,0).$$

The lower block performs division; it transforms

$$(q_i;\ y+1,z,x,0) \quad \text{into} \quad (q_j;\ y,z,0,x \text{ DIV } d),$$

where q_j depends on x MOD d. Notice that the non-zero content of one of the registers is (modified and) transferred from one register to the other. The opposite transfers (without modification) can be obtained by the choice $a = 1$, $b = 0$, $d = 1$.

The third stage need not depend on f (similarly as the first stage), and the block for it is given in Fig. 5; macroinstructions are not used. The block transforms

$$(q_{24};\ 0, z, 0, 2^x) \quad \text{into} \quad (q_0;\ x, z, 0, 0).$$

However, if we remove the bullet under the vertical arrow from q_{30} (and if we direct the arrow back to q_{30}) then 2^x can be replaced by arbitrary $2^x \cdot (2y+1)$ in the formula above. (In other words, the block computes the exponent of 2 in the factorization of R_3.) Table 2 contains the computation for $2^x = 4$. □

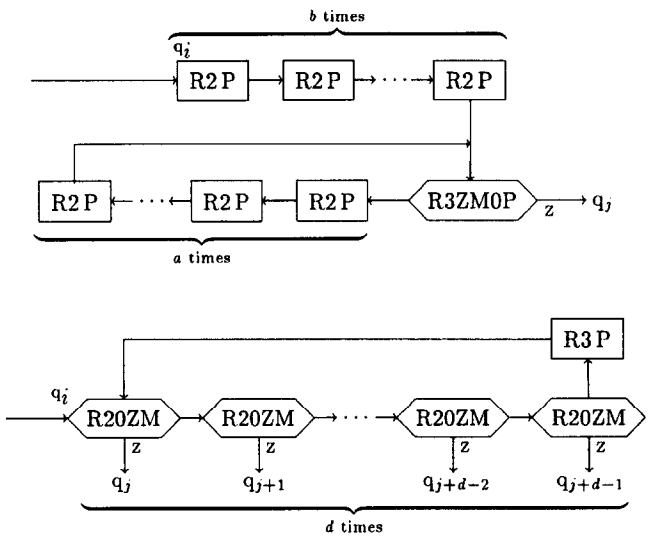

Fig. 4. Multiplication and division by constants in R3a-machines.

Table 2
The computation of the block from Fig. 5 for $x = 4$

0: (q24; 0, 9, 0, 4)	12: (q21; 2, 9, 2, 1)	24: (q23; 3, 9, 0, 0)
1: (q25; 0, 9, 0, 3)	13: (q22; 2, 9, 1, 1)	25: (q21; 3, 9, 1, 1)
2: (q29; 0, 9, 0, 2)	14: (q23; 2, 9, 0, 1)	26: (q22; 3, 9, 0, 1)
3: (q28; 0, 9, 1, 3)	15: (q21; 2, 9, 1, 2)	27: (q24; 2, 9, 0, 1)
4: (q24; 0, 9, 1, 2)	16: (q22; 2, 9, 0, 2)	28: (q25; 2, 9, 0, 0)
5: (q25; 0, 9, 1, 1)	17: (q24; 1, 9, 0, 2)	29: (q30; 3, 9, 0, 0)
6: (q29; 0, 9, 1, 0)	18: (q25; 1, 9, 0, 1)	30: (q31; 2, 9, 0, 0)
7: (q28; 0, 9, 2, 1)	19: (q29; 1, 9, 0, 0)	31: (q32; 2, 9, 1, 1)
8: (q24; 0, 9, 2, 0)	20: (q28; 1, 9, 1, 1)	32: (q33; 2, 9, 0, 1)
9: (q26; 1, 9, 2, 0)	21: (q24; 1, 9, 1, 0)	33: (q0; 2, 9, 0, 0)
10: (q27; 1, 9, 1, 0)	22: (q26; 2, 9, 1, 0)	
11: (q23; 2, 9, 1, 0)	23: (q27; 2, 9, 0, 0)	

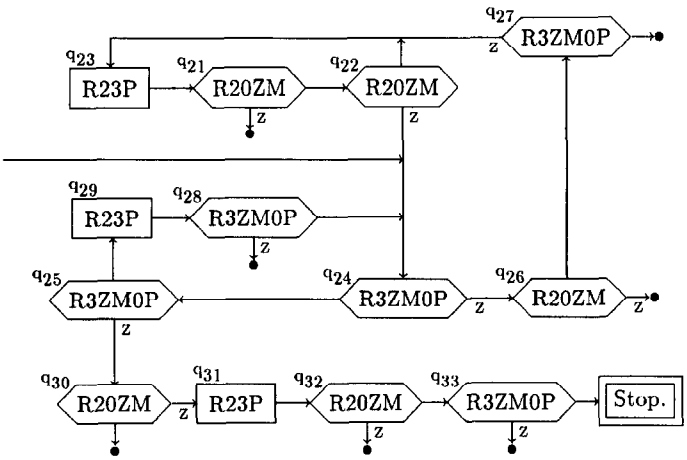

Fig. 5. The block of R3a-machines for $\log_2 x$.

5. The activity of the machine U_{32}

The role of the R-machine U_{32} is to simulate (some) R3a-machines; to speak about, an enumeration of R3a-machines must be introduced. Analogously, as R3a-machines in general, their presented enumeration has the only role in essential, to enable us to prove the Main Theorem. The enumeration will be chosen so that the number of instructions necessary to decode it is minimalized. Neither the elegance nor the computation time will be so important. In the following definition we shall define an R3a-machine M_x to every nonnegative integer x, and we shall define a set W of "good" numbers x; in what follows mainly "good" numbers will be considered.

Definition 5.1. (i) For every $i, j \in \mathbb{N}$, $z \in \{0,1,2\}$ we denote

$$(5.1.1)\quad Instr(i,3j+z) = \begin{cases} (\mathrm{q}_i, \langle \boxed{\text{R20ZM}} \rangle, \mathrm{q}_j, \mathrm{q}_{j+1}) & \text{if } z = 0; \\ (\mathrm{q}_i, \langle \boxed{\text{R3ZM0P}} \rangle, \mathrm{q}_j, \mathrm{q}_{j+1}) & \text{if } z = 1; \\ (\mathrm{q}_i, \boxed{\text{R23P}}, \mathrm{q}_j) & \text{if } z = 2. \end{cases}$$

(ii) For every $x \in \mathbb{N}$ we denote by Q_x the smallest set with the properties:
(1) $1 \in Q_x$;
(2) if $i \in Q_x$ and $\mathbf{F}(x,i) \geqslant 3$ then $\mathbf{F}(x,i)$ DIV $3 \in Q_x$;
(3) if $i \in Q_x$ and $\mathbf{F}(x,i)$ MOD $3 \neq 2$ then $(\mathbf{F}(x,i)$ DIV $3) + 1 \in Q_x$.

(iii) For every $x \in \mathbb{N}$ we denote

$$M_x = \{Instr(i, \mathbf{F}(x,i)) \mid i \in Q_x\}.$$

(iv) We denote by W the set of all $x \in \mathbb{N}$ such that for every $y \in \mathbb{N}$ and every configuration $(q_i; u, x, v, w)$, $i \neq 0$ in the computation of M_x from the initial configuration

$(q_1; 0, x, y, 0)$ the following condition is satisfied:

(5.1.2) $(\mathbf{F}(x,i) = 1 \wedge w > 0) \vee \mathbf{F}(x,i) = 2 \vee \mathbf{nd}(x+1,i) - \mathbf{F}(x,i) < \mathbf{nd}(x+1,1).$

Since $\mathbf{F}(x,i)$ is always positive the machine M_x cannot contain any instruction $(q_i; \langle\boxed{\text{R20ZM}}\rangle, q_0, q_1)$; this was the reason for the condition (3) in Definition 4.1. This condition is not too restrictive because we can use the macroinstruction GOTO q_0 for the end of computation. For every x we can consider the flowchart of "an infinite machine" $\{Instr(i, \mathbf{F}(x,i)) \mid i \in \mathbb{N} \setminus \{0\}\}$. Then Q_x is the set of all non-final inner states which can be reached from q_1 by the paths along the rows in this infinite flowchart (without paying attention to the labels 'z'). The set Q_x is finite because for every $j \in Q_x$ we have

$$j \leqslant (\mathbf{F}(x,y) \text{ DIV } 3) + 1 \leqslant (x+1) \text{ DIV } 3 + 1 = (x+4) \text{ DIV } 3;$$

we shall see on examples that the presented estimation is very rough. Generally speaking, not all q_i, $i \in Q_x$ occur in the normal computations of M_x. However, the sets Q_x^* of the subscripts of all reachable inner states cannot be effectively determined (in other words, it is undecidable whether an inner state occurs in a computation of M_x). The sets Q_x can be effectively determined.

W is the set of those x for which the machine U_{32} properly simulates M_x provided x is put into R_1, etc. (Of course, also for $x \notin W$ the machine U_{32} with x in R_1 simulates a machine, but the simulated machine can be distinct from M_x. We could analyze the activity of U_{32} in more general situations, and obtain more adequate definition of M_x. However, for our purposes $x \in W$ will be sufficient.) Notice that the set W is not recursive; we shall (implicitly) work with some recursive subsets of W.

Theorem 5.2. *For every $x \in W$ and all $y \in \mathbb{N}$ we have*

(5.2.1) $\Phi^2_{U_{32}}(x,y) = \Phi^1_{M_x}(y).$

Proof. We shall analyze the activity of the machine U_{32}, and we shall see that (under given assumptions) the machine U_{32} simulates the computation of M_x. So the required statement will be proved.

Let us enumerate the instruction codes of R3a-machines as follows:

(5.2.2) $\mathbf{n}(\langle\boxed{\text{R20ZM}}\rangle) = 0, \quad \mathbf{n}(\langle\boxed{\text{R3ZM0P}}\rangle) = 1, \quad \mathbf{n}(\boxed{\text{R23P}}) = 2;$

that corresponds to the formula (5.1.1). Let the simulation step for an instruction (q_i, X, q_j) of M_x ought to be performed.

The machine U_{32} starts the simulation either in the inner state q_{15} or in q_1; in the first case R_4 contains i, in the second case it contains $i-1$. The registers R_0, R_2, R_3 (now and always) correspond to the same registers of M_x. The register R_1 contains the number x; this number will be preserved as the sum $R_1 + R_7$ during the

whole simulation. The sum R_5+R_6 must be sufficiently small, more precisely less than $\mathbf{nd}(x+1,1)$. The register R_5 is always empty at this stage. However, the content of R_6 is equal to 0 only in the first step; later it may be positive, but the requested bound is arranged by (5.1.2).

The simulation step consists of three phases which are performed by three parts of U_{32} shown in dotted rectangles (see Fig. 1).

(1) Computation of $3j+\mathbf{n}(X)$ is performed by the upper dotted rectangle, called also *the instruction reader*; the result is put into R_5, and R_4 becomes empty. This phase will be described in more details later.

(2) Computation of j and $\mathbf{n}(X)$ (division by 3 in essential) is performed by lower right-hand rectangle, *the decoder*. The quotient j is put into R_4, and R_5 becomes 0. The remainder (which determines the operation code by (5.1)) is not stored into a register but it determines one of the three output arrows.

(3) Immediate simulation steps are performed by the lower left-hand rectangle, *the simulation block*. Its left-hand and right-hand output arrows correspond to the next inner states q_{j+1} and q_j, respectively. If the right-hand output arrow is used then a test for halting is performed (see q_{32}) before the phase (1) of the next simulation step follows.

The phases 2 and 3 are clear. We give some more details to the phase 1. The value $\mathbf{F}(x,i)(=3j+\mathbf{n}(X))$ is computed so that $x+1$ is divided by consecutive integers, starting with 1 or R_5+R_6+1. Then $\mathbf{F}(x,i)$ will be the ith nonzero test. (Now the requested bound for R_5+R_6 is clear: the initial sum must be less than the least positive non-divisor of $x+1$. Then no non-divisor is lost.) One division is performed by the nine instructions in the three upper lines of Fig. 1. Technically, in each division x is moved from R_1 to R_7 at first, then the divisor is enlarged (and put into R_6), and only after that the proper division is performed, and x is moved from R_7 back to R_1. (During the division R_6 and R_7 are simultaneously diminished and R_5 is raised. The actual divisor is preserved as R_5+R_6. The content of R_6 is renewed from R_5 whenever necessary. The quotient is not computed; we need only the remainder which arises in R_5.) Divisors are skipped (the test in q_{13}), every nondivisor decreases the content of R_4. When R_4 become empty the phase (1) is finished.

Now pay attention to the formula (5.1.2). If the first or the second member is valid then the simulated step is the last one (hence the instruction reader will work no more). Otherwise the third member expresses that the content of R_6 (equal to $\mathbf{nd}(x+1,i)-\mathbf{F}(x,i)$) is sufficiently small for correct work of the instruction reader in the next simulation step.

The above described simulating cycle is performed for every step of the computation of M_x. Notice that the computation of U_{32} starts from q_1 and with $R_4=0$. Hence the instruction of M_x with the first component q_1 is simulated. Both U_{32} and M_x use q_1 as the initial inner state.) The halting of M_x is checked by the instruction q_{32} of U_{32}, and the result can be read in R_0. □

Table 3
A computation of U_{32}

0: (q1; 0, 23, 1, 0, 0, 0, 0, 0)	936: (q3; 0, 22, 1, 0, 0, 3, 0, 0)
1: (q2; 0, 23, 1, 0, 0, 0, 0, 0)	1002: (q3; 0, 0, 1, 0, 0, 3, 0, 22)
2: (q3; 0, 22, 1, 0, 0, 0, 0, 0)	1003: (q1; 0, 0, 1, 0, 0, 3, 0, 23)
68: (q3; 0, 0, 1, 0, 0, 0, 0, 22)	1004: (q6; 0, 0, 1, 0, 0, 3, 0, 23)
69: (q1; 0, 0, 1, 0, 0, 0, 0, 23)	1227: (q13; 0, 23, 1, 0, 0, 4, 0, 0)
70: (q6; 0, 0, 1, 0, 0, 0, 0, 23)	1228: (q1; 0, 23, 1, 0, 0, 4, 0, 0)
71: (q4; 0, 0, 1, 0, 0, 0, 1, 23)	1297: (q1; 0, 0, 1, 0, 0, 4, 0, 23)
72: (q7; 0, 0, 1, 0, 0, 0, 1, 23)	1298: (q6; 0, 0, 1, 0, 0, 4, 0, 23)
73: (q8; 0, 0, 1, 0, 0, 0, 1, 23)	1535: (q29; 0, 23, 1, 0, 1, 0, 1, 0)
74: (q9; 0, 0, 1, 0, 0, 0, 0, 23)	1536: (q1; 1, 23, 1, 0, 1, 0, 1, 0)
75: (q10; 0, 0, 1, 0, 0, 1, 0, 23)	1605: (q1; 1, 0, 1, 0, 1, 0, 1, 23)
76: (q11; 0, 0, 1, 0, 0, 1, 0, 23)	2728: (q1; 1, 23, 1, 0, 0, 4, 1, 0)
77: (q12; 0, 0, 1, 0, 0, 1, 0, 22)	2797: (q1; 1, 0, 1, 0, 0, 4, 1, 23)
319: (q12; 0, 22, 1, 0, 0, 1, 0, 0)	3015: (q1; 1, 23, 1, 0, 0, 6, 0, 0)
320: (q7; 0, 23, 1, 0, 0, 1, 0, 0)	3084: (q1; 1, 0, 1, 0, 0, 6, 0, 23)
327: (q9; 0, 23, 1, 0, 0, 0, 0, 0)	3317: (q14; 1, 23, 1, 0, 0, 3, 4, 0)
328: (q10; 0, 23, 1, 0, 0, 1, 0, 0)	3318: (q16; 1, 23, 1, 0, 0, 3, 4, 0)
329: (q13; 0, 23, 1, 0, 0, 1, 0, 0)	3327: (q24; 1, 23, 1, 0, 1, 0, 4, 0)
330: (q1; 0, 23, 1, 0, 0, 1, 0, 0)	3328: (q32; 1, 23, 0, 0, 1, 0, 4, 0)
399: (q1; 0, 0, 1, 0, 0, 1, 0, 23)	3329: (q15; 1, 23, 0, 0, 1, 0, 4, 0)
400: (q6; 0, 0, 1, 0, 0, 1, 0, 23)	3330: (q1; 1, 23, 0, 0, 0, 0, 4, 0)
401: (q4; 0, 0, 1, 0, 0, 1, 1, 23)	3400: (q6; 1, 0, 0, 0, 0, 0, 4, 23)
409: (q11; 0, 0, 1, 0, 0, 1, 1, 23)	3619: (q21; 1, 23, 0, 0, 0, 2, 1, 0)
410: (q12; 0, 0, 1, 0, 0, 1, 1, 22)	3620: (q22; 1, 23, 0, 0, 0, 1, 1, 0)
630: (q12; 0, 22, 1, 0, 0, 1, 1, 0)	3623: (q18; 1, 23, 0, 0, 1, 0, 1, 0)
631: (q7; 0, 23, 1, 0, 0, 1, 1, 0)	3624: (q27; 1, 23, 0, 0, 1, 0, 1, 0)
632: (q8; 0, 23, 1, 0, 0, 1, 1, 0)	3625: (q29; 1, 23, 0, 0, 1, 0, 1, 0)
633: (q9; 0, 23, 1, 0, 0, 1, 0, 0)	3626: (q1; 2, 23, 0, 0, 1, 0, 1, 0)
634: (q10; 0, 23, 1, 0, 0, 2, 0, 0)	3695: (q1; 2, 0, 0, 0, 1, 0, 1, 23)
635: (q13; 0, 23, 1, 0, 0, 2, 0, 0)	4818: (q1; 2, 23, 0, 0, 0, 4, 1, 0)
636: (q1; 0, 23, 1, 0, 0, 2, 0, 0)	4819: (q2; 2, 23, 0, 0, 0, 4, 1, 0)
637: (q2; 0, 23, 1, 0, 0, 2, 0, 0)	4820: (q3; 2, 22, 0, 0, 0, 4, 1, 0)
638: (q3; 0, 22, 1, 0, 0, 2, 0, 0)	4886: (q3; 2, 0, 0, 0, 0, 4, 1, 22)
704: (q3; 0, 0, 1, 0, 0, 2, 0, 22)	4887: (q1; 2, 0, 0, 0, 0, 4, 1, 23)
705: (q1; 0, 0, 1, 0, 0, 2, 0, 23)	5105: (q1; 2, 23, 0, 0, 0, 6, 0, 0)
706: (q6; 0, 0, 1, 0, 0, 2, 0, 23)	5106: (q2; 2, 23, 0, 0, 0, 6, 0, 0)
719: (q12; 0, 0, 1, 0, 0, 1, 2, 22)	5107: (q3; 2, 22, 0, 0, 0, 6, 0, 0)
928: (q12; 0, 22, 1, 0, 0, 2, 1, 0)	5173: (q3; 2, 0, 0, 0, 0, 6, 0, 22)
929: (q7; 0, 23, 1, 0, 0, 2, 1, 0)	5174: (q1; 2, 0, 0, 0, 0, 6, 0, 23)
930: (q8; 0, 23, 1, 0, 0, 2, 1, 0)	5414: (q22; 2, 23, 0, 0, 0, 0, 4, 0)
931: (q9; 0, 23, 1, 0, 0, 2, 0, 0)	5420: (q2; 1, 23, 0, 0, 1, 0, 4, 0)
932: (q10; 0, 23, 1, 0, 0, 3, 0, 0)	5487: (q3; 1, 0, 0, 0, 1, 0, 4, 22)
933: (q13; 0, 23, 1, 0, 0, 3, 0, 0)	5488: (q1; 1, 0, 0, 0, 1, 0, 4, 23)
934: (q1; 0, 23, 1, 0, 0, 3, 0, 0)	5704: (q1; 1, 23, 0, 0, 0, 4, 1, 0)
935: (q2; 0, 23, 1, 0, 0, 3, 0, 0)	8680: (q1; 1, 23, 0, 0, 0, 4, 1, 0)

A computation of the machine U_{32} is shown in Table 3. To show a bigger piece, many configurations are deleted; deleted parts can be recognized by the number of steps. The last two displayed configurations show that the computation is ultimately periodic and its periodic and its period divides 2976 (it is equal to 2976 indeed).

6. Existence of "good" R3a-Machines

To finish the proof of the part (a1) of the Main Theorem we have to prove that every unary partial recursive function is computable by an R3a-machine with "good" number. It will be done below.

At first we shall explain the principle how Lemma 3.2 can be used to find the numbers of (some) R3a-machines. Let all non-final inner states of an R3a-machine M belong to $q_1, q_2, \ldots, q_k$. We shall look for an integer x such that for every instruction (q_i, X, q_j, q_{j+1}) (without the fourth component if $X = \boxed{\text{R23P}}$) of M it holds

$$\mathbf{F}(x, i) = 3j + \mathbf{n}(X); \tag{6.1}$$

remember that X, j are uniquely determined by i and that $\mathbf{n}(X)$ was defined in (5.2.2). If the right-hand sides of (6.1) are positive (and we may restrict ourselves to such machines) we can apply Lemma 3.2. The integer x given by the lemma is not a number of M; however, (the reachable part of) the flowchart of M can be embedded into the flowchart of M_x. This is only a simplified principle, in fact some further technical problems must be solved to arrange $x \in W$. (Most of them could be avoided if we allow arbitrary initial states instead of q_1, or if we add two new instructions into U_{32}, which will empty the register R_6 before leaving the instruction reader.) For that we shall need the following result.

Lemma 6.1. *For every $c > 1, k > 1$ there are $s > c$ and primes*

$$r_1 < r_2 < \ldots, r_{s-1} < r_s = p_0 < p_1 < \cdots < p_k \tag{6.1.1}$$

such that if we denote $h(t) = \sum_{j=1}^{s} \left\lfloor \frac{t}{r_j} \right\rfloor + \sum_{i=1}^{s} \left\lfloor \frac{t}{p_i} \right\rfloor$ then

$$r_1^2 > p_k; \tag{6.1.2}$$

$$h(p_k) = h(p_0) + k; \tag{6.1.3}$$

$$c \cdot h(p_k) < p_0 - c; \tag{6.1.4}$$

$$p_k - r_1 < c \cdot h(p_0). \tag{6.1.5}$$

$$r_{j+1} - r_j < r_1 - c \quad \textit{for all } j = 1, \ldots, s-1; \tag{6.1.6}$$

$$c \cdot h(r_{j+1}) < r_j - c \quad \textit{for all } j = 1, \ldots, s-1. \tag{6.1.7}$$

The lemma will be proved later, now we only explain its meaning. It will help us to find the good number x of an R3a-machine (equivalent with) M. The integer k will be the number of its instructions and $c = 3$ corresponds to the number of input arrows of the simulation block of U_{32}. The primes $p_1, \ldots, p_k$ will be the non-divisors of $x+1$ corresponding to the non-final inner states of M. Lemma 6.1 summarizes conditions on non-divisors of $x + 1$, without explicit dealing with x. (Through the function h it

deals implicitly with all non-divisors of x up to p_k, even if it explicitly mentions only prime nondivisors.)

Theorem 6.2. *For every R3a-machine M there is $x \in W$ such that $\phi_M = \phi_{M_x}$.*

Proof. The flowchart of M_x will be obtained by joinng many macroinstructions GOTO (see Fig. 3) before the starting state of M. The macroinstructions obviously do not influence the result of the computation but are necessary in the construction of x. We shall assume that M is reduced in the sense that all its instructions are reachable along the arrows from the initial instructions (it does not mean that they are really used in some normal computations).

Let the machine M use (some of) the inner states $q_0, q_1, \ldots, q_k$. Let us choose a system of primes (6.1.1) which satisfy all the conditions from Lemma 6.1. We shall also use the function h. Let Q denote the set of all primes $p \leqslant p_k$ which do not occur in (6.1.1). We shall write a system of congruences for $x+1$. It will consist of four parts. The first part will be

$$(6.2.1) \quad x+1 \equiv 0 \,(\mathrm{mod}\, p^e), \quad \text{where } e = \left\lfloor \frac{\ln p_k}{\ln p} \right\rfloor \quad \text{for all primes } p \in Q.$$

The other parts will be written later; they will imply

$$(6.2.2) \quad x+1 \not\equiv 0 \,(\mathrm{mod}\, p) \quad \text{for all primes } p \text{ from } (6.1.1).$$

(6.1.2), (6.2.1) and (6.2.2) imply that for all positive $\{y \leqslant p_k\}$

$$y \nmid (x+1) \Leftrightarrow \mathrm{MaxPr}(y) \in \{r_1, \ldots, r_s, p_1, \ldots, p_k\}.$$

Let us denote $d = h(r_s) = h(p_0)$. The condition (6.1.2) arranges that every positive integer $y \leqslant p_k$ has at most one divisor among (6.1.1). Hence, $h(y)$ is the number of non-divisors of $x+1$ which do not exceed y. Therefore, we have

$$(6.2.3) \quad \mathbf{nd}(x+1, h(y)) = y \quad \text{for every } y \leqslant p_k, y \nmid x+1,$$

in particular,

$$(6.2.4) \quad \mathbf{nd}(x+1,\ d+i) = p_i \quad \text{for all } i = 0, 1, \ldots, k.$$

For every $i = 1, \ldots, k$ the prime p_{d+i} (i.e. the $(d+i)$th non-divisor of $x+1$) will correspond to the inner state q_i (but p_0 do not correspond to the final inner state q_0). If M contains $Instr(i,\ 3j+z)$, $j > 0$ then M_x will contain $Instr(i+d,\ 3j+3d+z)$; for $j = 0$ the member $3d$ is not added. The corresponding congruences are

$$(6.2.5) \quad x+1 \equiv 3j + \mathrm{sg}(j) \cdot 3d + z \,(\mathrm{mod}\, p_i) \quad \text{if } Instr(i, 3j+z) \in M,$$

where sg denotes the signum function, sg(0)=0, $\mathrm{sg}(j) = 1$ for $j > 0$.

Finally, M_x will contain several GOTO macroinstructions to reach the inner state q_{d+1} from the initial state q_1. To arrange that, let us choose a sublist

$$(6.2.6) \quad q_1 = r_1, q_2, \ldots, q_t = p_0, q_{t+1} = p_1$$

of (6.1.1) such that q_{j+1} is reachable from q_i, i.e.

$$q_i - r_1 < 3h(q_{i+1}) < q_i - 3 \quad \text{for all } i = 1, \ldots, t.$$

(The symbols q_i and q_i must be distinguished; there is no direct relationship between them.) The list (6.2.6) can easily be constructed backwards; we know q_t (even if we do not know t). If we know $q_{j+1} = r_i, i > 1$, we can compute $h(q_{j+1})$, and then choose a suitable q_j among $r_1, \ldots, r_{i-1}$. The right-hand inequality follows from (6.1.7). We finish when $q_1 = r_1$ is chosen (only at this moment t is determined). The process is possible by (6.1.6) and (6.1.7). The number of chosen q_i can be controlled in some degree, and so the length of (6.2.6) modulo 3 can be prescribed; let $3|t$. The corresponding congruences will be

$$(6.2.7) \quad x + 1 \equiv 3h(q_{i+1}) + (i+1) \text{ MOD } 3 \pmod{q_i} \text{ for all } i = 1, \ldots, t.$$

Finally, we added e.g. the congruences

$$(6.2.8) \quad x + 1 \equiv 2 \pmod{p} \quad \text{for all other primes } p < p_k;$$

the words "for all others" mean "not used in (6.2.1), (6.2.5), (6.2.7)". (Of course, for practical computation it would be better to leave here $\not\equiv 0$ instead of $\equiv 2$; a smaller solution x can be obtained.)

If x is a (positive) solution of the system of congruences (6.2.1), (6.2.5), (6.2.7), (6.2.8) then the machine M_x has all required properties, and is equivalent with M. (In fact, the flowchart of M with additional GOTO instructions is embedded into the flowchart of M_x.) However, the moduli of the system of congruences are pairwise relatively prime, and hence the system is solvable by Chinese Remainder Theorem.

□

Now the part (a1) of the Main Theorem can be obtained as follows. By Theorem 4.2 every unary partial recursive function f is computable by an R3a-machine M. By Theorem 6.2 we can replace M by M_x for some $x \in W$. Finally, by Theorem 5.2 M_x is simulated by U_{32}.

Proof of Lemma 6.1. We choose sufficiently large N so that:

(1) For every X, $N^{0.6} \leqslant X \leqslant N$ the interval $(X, X + N^{0.56})$ contains more than $N^{0.01}$ primes.

(2) The set $\{x \leqslant N | \mathrm{MaxPr}(x) > N^{0.6}\}$ has at least $N/c + N^{0.9}$ elements.

(3) For every X, $N^{0.6} \leqslant X \leqslant N$ the interval $(X, X + N^{0.57})$ contains a subinterval of length at most $N^{0.01}$ which contains at least $k + 1$ primes.

Condition (1) follows from the second formula of Theorem 3.3; we can divide the mentioned interval into $N^{0.01}$ intervals of length $N^{0.55}$, and each of them contains (many) primes. Condition (2) can be obtained by Theorem 3.4 because $\ln \frac{1}{0.6} = 0.5108\ldots > \frac{1}{2} \geqslant \frac{1}{c}$. Condition (3) can be obtained by Theorem 3.3; we use that $\frac{13}{23} = 0.5625\ldots < 0.57$. The exponent 0.01 is used instead of arbitrarily small positive ε; it

suffices here. (Notice that for $c > 2$ we can replace the exponents 0.6, 0.56, etc. by bigger ones, and so use a little weaker number theoretical results.)

Our aim is to choose (6.1.1) so that $p_0, p_1, \ldots, p_k$ will be consecutive primes from the interval $(N^{0.6}, N)$ such that $p_k - p_0 \leqslant N^{0.01}$ and

$$\{r_1, \ldots, r_s\} = \left\{ p \in (N^{0.6}, N) \mid p \text{ prime} \wedge \left\lfloor \frac{p_0}{p} \right\rfloor = \left\lfloor \frac{p_k}{p} \right\rfloor \right\}.$$

So (6.1.2), (6.1.3) would be obviously satisfied. Also (6.1.6) can be proved. Indeed, let R denote the set of primes p, $N^{0.6} \leqslant p \leqslant p_k$ which do not belong to (6.1.1); we also have

$$R = \{\text{MaxPr}(x) \mid x \in (p_0, p_k)\} \cap (N^{0.6}, p_0).$$

If (6.1.6) does not hold then we have $r_{j+1} - r_j > r_1 - c > N^{0.6}$ and, hence, by (1) the interval (r_j, r_{j+1}) contains at least $N^{0.05}$ primes. All these primes beong to R, which contradicts $\text{card}(R) < p_k - p_0 \leqslant N^{0.01}$.

Now let us denote $g(t) = \sum_{N^{0.6} < p \leqslant N} \lfloor \frac{t}{p} \rfloor$. Let $g^{-1}t$ be defined as the minimal y such that $g(y) = t$, and analogously for h. While the function h depends on (6.1.1) (and so indirectly on N), the function g depends on N only. However, h and g are near each to the other in the sense

$$g(t) - N^{0.41} \leqslant h(t) \leqslant g(t).$$

Indeed, the difference between $g(t)$ and $h(t)$ is caused only by the primes from R, and therefore

$$0 \leqslant g(t) - h(t) = \sum_{p \in R} \left\lfloor \frac{t}{p} \right\rfloor < \sum_{t \in R} \frac{t}{N^{0.6}} < \text{card}(R) \cdot N^{0.4} \leqslant N^{0.41}.$$

To arrange (6.1.4) and (6.1.7) we must choose a suitable $p_k \in (N^{0.6}, N)$; by this choice (and the decisions above) the list (6.1.1) will be completely determined. Let q be the smallest prime from this interval such that the interval $(q, g^{-1}\lfloor (q-c)/c \rfloor)$ contains less than k primes. Such q exists because $g^{-1}\lfloor (q-c)/c \rfloor - q$ is greater than $N^{0.6}$ for small q (e.g. near to $2N^{0.6}$) and negative for big q (near to N); moreover, the function $G(t) = g^{-1}\lfloor \frac{t}{c} \rfloor - t$ cannot *decrease* too quickly. Now we can apply (3) to the interval $(q - N^{0.57}, q)$, and choose $p_0, \ldots, p_k$ from it so that $p_k - p_0 < N^{0.01}$.

Then we have

$$\begin{aligned} h^{-1}\left\lfloor \frac{p_k - r_1}{c} \right\rfloor &\leqslant h^{-1}\left\lfloor \frac{q - \lfloor N^{0.6} \rfloor}{c} \right\rfloor \leqslant g^{-1}\left\lfloor \frac{q - \lfloor N^{0.6} \rfloor}{c} \right\rfloor + N^{0.41} \\ &\leqslant g^{-1}\left\lfloor \frac{q-c}{c} \right\rfloor - N^{0.6} + N^{0.41} < q - \frac{1}{2}N^{0.6} < p_0. \end{aligned}$$

Therefore, we have also $h^{-1}\lfloor (p_k - r_1)/c \rfloor < p_0$, what can be transformed into (6.1.5).

□

Primes seem to be distributed much more regularly than Theorem 3.3 shows. For example, it seems that

$$\lim_{N\to\infty}(\pi(N+N^{0.01})-\pi(N))=\infty.$$

Hence, we can take arbitrary sufficiently large prime as p_0, then determine $p_1,\ldots,p_k$ as the consecutive primes, and computer r_i backwards from $r_s=p_0$ (without knowing s in advance). Also r_i will be consecutive primes whenever possible; however, some gaps are necessary to obtain (6.1.3). We can also replace p_0 by p_1 in some places of Lemma 6.1. (Roughly speaking, we need not jump from p_0 to p_n and back, but only from p_0 and p_n to p_1 and from p_1 to p_n. For general considerations p_1 was eliminated from the inequalities of Lemma 6.1 to simplify their form.)

The obtained numbers of machines are huge, even if we follow only the idea of the construction above, and pay not any attention to the numerical bounds requested in number-theoretical theorems ([9] does not mention explicitly these bounds; maybe they are not effective). Notice that we need not use Lemma 6.1 or all inequality conditions above in particular cases when suitable systems of primes are determined explicitly (and their required properties are verified by computations). We shall show some numerical examples.

Example 6.3. For $c=3$, $k=6$ and $p_0\approx 590$ a computer computation gives e.g. the following possibility:

(a) $p_1,\ldots,p_6$ =593, 599, 601, 607, 613, 617; further we have $p_0=587$ and $h(p_i)=190+i$ for $i=0,1,\ldots,6$.

(b) There are 9 divisors of integers 588, 589,...,616 among the primes between 52 and 588; they are 59, 61, 67, 101, 149, 151, 197, 199, 307.

(c) The other $s=83$ primes are 53, 71, 73,..., 577, 587; they are taken as r_i.

(d) A possible choice of q_j starts 53, 71, 97, 137,...; However, the second or the third member (or both) can be deleted; so we can arrange $3|t$. Without the last condition we can arrange arbitrary t, $10\leqslant t\leqslant 23$.

(e) The moduli p^e of the congruences (6.2.1) will be

$$2^9,\ 3^5,\ 5^3,\ 7^3,\ 11^2,\ 13^2,\ 17^2,\ 19^2,\ 23^2,\ 29,\ 31,\ 37,\ 41,\ 43,\ 47$$

and 9 primes listed in (b) (together 24 values).

A trivial bound for x is given by the product of all considered moduli. Practically, it can be diminished by suitable choice of non-fixed rests by some moduli, but x obviously remains very large.

Example 6.4. If we consider $c=3$ and choose $p_0=9973$ (the greatest prime under 10000) then we may take $k=22$ (and any less value, of course). Then $p_1,\ldots,p_{22}$ will be the primes from 10007 to 10181. It will hold $h(p_i)=3285+i$. Ther will be 1077 primes r_i, from $r_1=331$ to $r_{1077}=9973$; the number of excluded primes will be

86 (the least one is 337, which is excluded becaused 337=MaxPr(10110)), and $t = 15$ can be arranged. 86 prime powers with bases less than 331 are considered.

We shall give also examples for two other values of c; one of them, $c = 2$, is necessary for the part (b4); the value $c = 4$ is only as an illustrative example.

Example 6.5. If we consider $c = 2$ and choose $p_0 = 8999$ (it appeared better than 9973) then we may take $k = 9$. We shall have $p_1 = 9001$, $p_9 = 9059$ and $h(p_i) = 4474 + i$. There will be $s = 1054$ primes r_i, from $r_1 = 149$ to $r_{1054} = 8999$; the number of excluded primes will be 29 (the least one is 167, which is excluded because 167=MaxPr(9018), and the greatest is 4523), and we can arrange $t = 24$. Besides, we have to consider 34 prime power divisors with the bases less than 149.

Example 6.6. If we consider $c = 4$ and choose $p_0 = 9973$ (the same as in Ex. 6.4) then we may take $k = 42$. We shall have $p_1 = 10007$, $p_{42} = 10357$ and $h(p_i) = 2447 + i$. There will be $s = 1002$ primes r_i, from $r_1 = 577$ to $r_{1002} = 9973$; the number of excluded primes will be 122 (the least one is 587, the greatest 5171), and we can arrange $t = 12$. Further, 105 prime powers with bases less than 577 are considered.

7. The machines derived from U_{32}

The machines U_{22}, U_{21}, U_{29}, U_{20}, U_{19} for the parts (a2),...,(b3) of the Main Theorem will work similarly as the machine U_{32}. (The subscripts again denote numbers of instructions.) In particular, the partition into three dotted block and their role will be preserved. Therea are two ways how the number of instructions is diminished:

(1) The instructions $\langle Ri \rangle$ and $\boxed{Ri\,M}$ are glued together whenever possible.

(2) The number of registers of simulated machines is reduced to 2. To do that, we shall simulate R2-machines from 2.2.11. In the proof we shall need Lemma 6.1 for $c = 2$ (while for U_{32} we have used $c = 3$, what is a weaker statement).

The first possibility is used in (a2) and (a3), the second one in (b1), both possibilities in (b2) and (b3). All these machines have very similar structure, and work very similarly to U_{32}. Therefore, we shall present only one of them. The machine U_{19} is presented in Fig. 6. The inner states are numbered so that the relationship with U_{32} is as clear as possible. From similar reason in U_{19} the roles of registers are preserved; only the register R_3 is not used at all. The machine U_{19} is not "nice" in the sense that the used instruction base is not minimal; $\langle Ri \rangle$ can be eliminated. However, to do so we need one instruction more: the unique $\langle R6 \rangle$ instruction must be replaced by $\langle \boxed{R6\,ZM} \rangle$ and $\boxed{R6\,P}$. So a nicer machine U_{20} is obtained. The situation between U_{21} and U_{22} is quite similar.

Remark. There is another not quite "honest" possibility to spare one instruction in any of the machines above: To allow the dynamical stop. (The spared instruction will

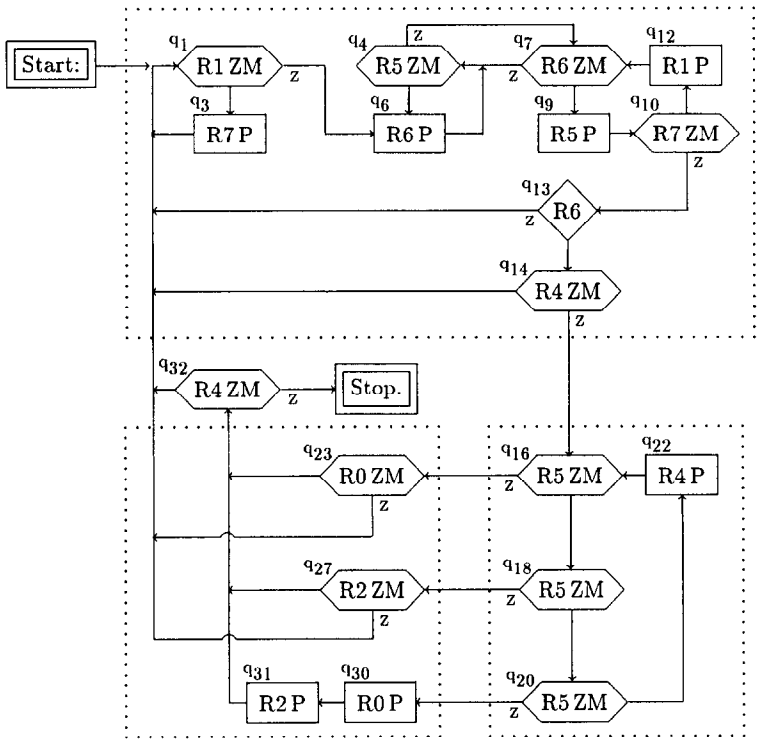

Fig. 6. The universal register machine U_{19}.

be q_{32}.) The machines will always work forever, but sometimes it will work in a trivial (and easily recognizable) cycle, and the content of the register R_0 will not be changed.

For an example of a small universal program we shall use the register programs from 2.2.5. To construct a small universal program from this class we shall linearly order (by the first component) the instructions of U_{32} and then we insert GOTO where necessary. GOTO can be simulated by the conditional jump with respect to a register which is empty at this moment. The simplest way is to use a new register, R_8 in our case; in the normal computations it will always remain empty. It suffice to insert together 10 instructions; we can find them in Table 4. Instead of our usual operation codes we use the operation symbols INCR, DECR and a BASIC-like form of conditional jump in the table (and also in Theorem 7.1). We preferred to insert new non-numerical labels (capitals $A,\ldots,J$) so that the relationship to U_{32} remains clear. The re-enumeration (necessary from the formal point of view) is straightforward. So we obtain:

Theorem 7.1. *There is a strongly universal register program with 42 instructions of the forms* INCR R_i, DECR R_i, *and* IF $R_i = 0$ GOTO n.

Table 4
A strongly universal register program

1: IF $R_1=0$ GOTO 6	*C*: IF $R_8=0$ GOTO 7	24: DECR R_2
2: DECR R_1	13: IF $R_6=0$ GOTO 1	*F*: IF $R_8=0$ GOTO 32
3: INCR R_7	14: IF $R_4=0$ GOTO 16	25: IF $R_0=0$ GOTO 32
A: IF $R_8=0$ GOTO 1	15: DECR R_4	26: DECR R_0
4: IF $R_5=0$ GOTO 7	*D*: IF $R_8=0$ GOTO 1	*G*: IF $R_8=0$ GOTO 1
5: DECR R_5	16: IF $R_5=0$ GOTO 23	27: IF $R_3=0$ GOTO 29
6: INCR R_6	17: DECR R_5	28: DECR R_3
B: IF $R_8=0$ GOTO 4	18: IF $R_5=0$ GOTO 27	*H*: IF $R_8=0$ GOTO 32
7: IF $R_6=0$ GOTO 4	19: DECR R_5	29: INCR R_0
8: DECR R_6	20: IF $R_5=0$ GOTO 30	*I*: IF $R_8=0$ GOTO 1
9: INCR R_5	21: DECR R_5	30: INCR R_2
10: IF $R_7=0$ GOTO 13	22: INCR R_4	31: INCR R_3
11: DECR R_7	*E*: IF $R_8=0$ GOTO 16	32: IF $R_4=0$ GOTO 33
12: INCR R_1	23: IF $R_2=0$ GOTO 25	*J*: IF $R_8=0$ GOTO 15

Of course, similar results could be obtained for some other classes of register programs. We shall not do that in the present paper.

8. A universal machine with 14 instructions $\langle \overline{\mathrm{R}i\mathrm{PZ} : \mathrm{R}k} \rangle$

To finish the proof of the Main Theorem it remains to prove the part (b4). The general schema of the proof remain similar to the above but there are more differences than there were, e.g., between (a1) and (b3). Some parts of the proof will be only sketched.

Proof of the part (b4). The machine U_{14} (Fig. 7) is divided into three blocks which have the same roles as those of U_{32}; we shall also use the same names for them. A small difference is that the instruction q_{14} (which decides halting) belong to the instruction reader. (Therefore, it is not clear whether "dynamical stop" allows to spare one instruction). To understand the activity of U_{14} we have to realize that

(1) $\langle \overline{\mathrm{R}i\mathrm{PZ} : \mathrm{R}i} \rangle$ is equivalent with $\boxed{\mathrm{R}i := 0}$ and

(2) $\langle \overline{\mathrm{R}i\mathrm{PZ} : \mathrm{R}k} \rangle$ is equivalent with $\boxed{\mathrm{R}i\,\mathrm{P}}$ provided we know $\mathrm{R}_i \neq \mathrm{R}_k$.

The second statement can be better explored if we restrict the analyse of U_{14} to the normal computations; then, e.g., we can assume that always $R_6 < R_1$ and $R_4 < R_1$. Further, it is clear from the flowchart that the content of R_1 remains constant in every computation. It will always contain the number of a simulated machine; in U_{32} we have to preserve the number of a simulated machine as the sum of two registers. Analogously, during a division the divisor permanently remains in one counter. The docoder divides (by 2) the difference $R_6 - R_5$, and not R_5. (There are small differences, e.g. ± 1 at some places. This fact must be considered but is not substantial.) A computation of U_{14} is presented in Table 5; we can see there the process of division very well. This remarks could suffice to understand the activity of the instruction reader and the decoder.

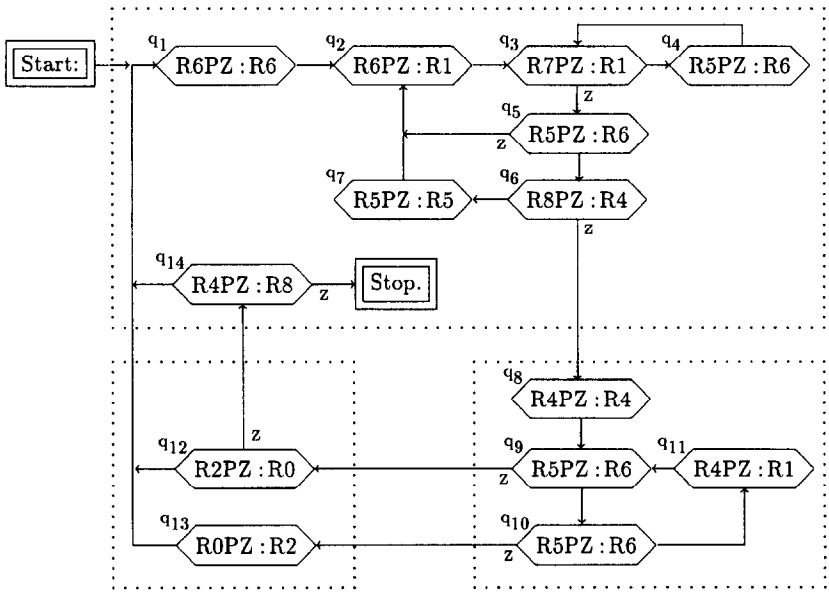

Fig. 7. The universal register machine U_{14}.

Table 5
A computation of U_{14}

0: (q1; 2, 5, 0, 0, 0, 0, 0, 0, 0,)	26: (q5; 2, 5, 0, 0, 0, 2, 2, 0, 0,)
1: (q2; 2, 5, 0, 0, 0, 0, 0, 0, 0,)	27: (q2; 2, 5, 0, 0, 0, 0, 2, 0, 0,)
2: (q3; 2, 5, 0, 0, 0, 0, 1, 0, 0,)	28: (q3; 2, 5, 0, 0, 0, 0, 3, 0, 0,)
3: (q4; 2, 5, 0, 0, 0, 0, 1, 1, 0,)	29: (q4; 2, 5, 0, 0, 0, 0, 3, 1, 0,)
4: (q3; 2, 5, 0, 0, 0, 1, 1, 1, 0,)	30: (q3; 2, 5, 0, 0, 0, 1, 3, 1, 0,)
5: (q4; 2, 5, 0, 0, 0, 1, 1, 2, 0,)	31: (q4; 2, 5, 0, 0, 0, 1, 3, 2, 0,)
6: (q3; 2, 5, 0, 0, 0, 0, 1, 2, 0,)	32: (q3; 2, 5, 0, 0, 0, 2, 3, 2, 0,)
7: (q4; 2, 5, 0, 0, 0, 0, 1, 3, 0,)	33: (q4; 2, 5, 0, 0, 0, 2, 3, 3, 0,)
8: (q3; 2, 5, 0, 0, 0, 1, 1, 3, 0,)	34: (q3; 2, 5, 0, 0, 0, 3, 3, 3, 0,)
9: (q4; 2, 5, 0, 0, 0, 1, 1, 4, 0,)	35: (q4; 2, 5, 0, 0, 0, 3, 3, 4, 0,)
10: (q3; 2, 5, 0, 0, 0, 0, 1, 4, 0,)	36: (q3; 2, 5, 0, 0, 0, 0, 3, 4, 0,)
11: (q4; 2, 5, 0, 0, 0, 0, 1, 5, 0,)	37: (q4; 2, 5, 0, 0, 0, 0, 3, 5, 0,)
12: (q3; 2, 5, 0, 0, 0, 1, 1, 5, 0,)	38: (q3; 2, 5, 0, 0, 0, 1, 3, 5, 0,)
13: (q5; 2, 5, 0, 0, 0, 1, 1, 0, 0,)	39: (q5; 2, 5, 0, 0, 0, 1, 3, 0, 0,)
14: (q2; 2, 5, 0, 0, 0, 0, 1, 0, 0,)	40: (q6; 2, 5, 0, 0, 0, 2, 3, 0, 0,)
15: (q3; 2, 5, 0, 0, 0, 0, 2, 0, 0,)	41: (q8; 2, 5, 0, 0, 0, 2, 3, 0, 0,)
16: (q4; 2, 5, 0, 0, 0, 0, 2, 1, 0,)	42: (q9; 2, 5, 0, 0, 0, 2, 3, 0, 0,)
17: (q3; 2, 5, 0, 0, 0, 1, 2, 1, 0,)	43: (q10;2, 5, 0, 0, 0, 3, 3, 0, 0,)
18: (q4; 2, 5, 0, 0, 0, 1, 2, 2, 0,)	44: (q13;2, 5, 0, 0, 0, 0, 3, 0, 0,)
19: (q3; 2, 5, 0, 0, 0, 2, 2, 2, 0,)	45: (q1; 2, 5, 0, 0, 0, 0, 3, 0, 0,)
20: (q4; 2, 5, 0, 0, 0, 2, 2, 3, 0,)	46: (q2; 2, 5, 0, 0, 0, 0, 0, 0, 0,)
21: (q3; 2, 5, 0, 0, 0, 0, 2, 3, 0,)	47: (q3; 2, 5, 0, 0, 0, 0, 1, 0, 0,)
22: (q4; 2, 5, 0, 0, 0, 0, 2, 4, 0,)	48: (q4; 2, 5, 0, 0, 0, 0, 1, 1, 0,)
23: (q3; 2, 5, 0, 0, 0, 1, 2, 4, 0,)	49: (q3; 2, 5, 0, 0, 0, 1, 1, 1, 0,)
24: (q4; 2, 5, 0, 0, 0, 1, 2, 5, 0,)	50: (q4; 2, 5, 0, 0, 0, 1, 1, 2, 0,)
25: (q3; 2, 5, 0, 0, 0, 2, 2, 5, 0,)	51: (q3; 2, 5, 0, 0, 0, 0, 1, 2, 0,)

In global, they use the same principle (and the same function **F**) as the corresponding blocks of the machine U_{32}. Moreover, some difficulties disappear because R_6 becomes empty always when the instruction reader starts to work. The activity of the simulation block is very simple; it simulates R2b-machines from 2.2.14. A little less trivial is the proof that R2b-machines compute sufficiently many partial recursive functions. Remember that R2b-machines use the instruction codes $\langle\overline{\text{R0PZ : R2}}\rangle$ and $\langle\overline{\text{R2PZ : R0}}\rangle$, their test instructions are restricted similarly as that in R3a-machines ($k_2 = k_1 + 1$ in (2.1)), and R_0 is their input and output register. We shall use the input coding function $h(x) = 2^x$. However, we cannot use the output decoding function $f(x) = \log_2(x)$; we have to use e.g. $\text{ex}_2(x)$ =the exponent of 2 in the factorization of x.

R3a-machines were able to multiply and to divide an integer (given in R_2) by arbitrary (positive) integer constants. Our R2b-machines are able to multiply, but are not able to divide (by integer constants > 1). However, they are able to multiply by *rational* constants $\frac{c}{d} > 1$. To do that, they start with x in R_0 and R_2 empty. Then in a cycle they add $c - d$ to R_0 and c to R_2 until equality takes place (and so R_2 becomes empty again; the equality stops to hold whenever it is observed). The exact output from the cycle enables to recognize x MOD d. So we can construct R2b-machines similarly as the second stage blocks of R3a-machines (the stage from 2^x to $2^{f(x)}$) but we shall use "a garbage prime" q greater than any constant used before. Divisions by d will be replaced by multiplication by $\frac{q}{d}$. So we can obtain the result of the form $2^{f(x)} \cdot q^{g(x)}$, and the function ex_2 can extract $f(x)$ from it. □

9. Strongly universal R-machines for *n*ary functions

Since n-ary functions can be coded by unary ones, universal machines for n-ary functions usually are not considered. For Turing machines a (nontrivial) coding is necessary, simply because they do not work with the numbers directly. However, for register machines the direct input and output of numbers is very natural; that was also an idea in the distinguishing of strong universality from the usual one. So it has sense to study the machines from the title of the paragraph.

Theorem 9.1. *There is a constant C such that for every n there is a strongly universal* R*-machine* U_n^* *for the set of n-ary partial recursive functions which has* $C+2n+3\lceil\sqrt{n}\rceil$ *instructions.*

Sketch of proof. The universal machine U_n^* will read the values from input registers at first, and code them into (say) two working registers. Only the block for reading input registers depends on n. For $n = 9$ it is displayed in Fig. 8; notice that every hexagon replaces two instructions. The other parts of U_n^* do not depend on n (up to the enumeration of registers). The input reading block works as follows. Let n input registers be placed into a $k \times k$ square schema (some positions may be empty), $k = \lceil\sqrt{n}\rceil$. The blcok obtains a question "which column" (from 0th to $(k-1)$th) in

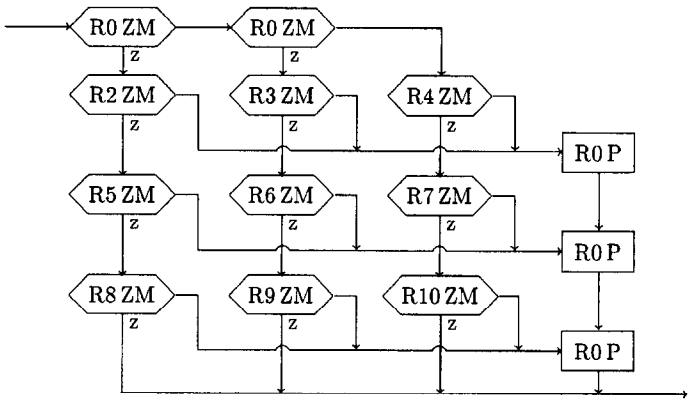

Fig. 8. The input reading block of U_n^* for $n = 9$.

the register R_0, and gives an answer "which row" (from the first to kth, enumerated from the bottom) also in R_0. The answer 0 shows that all input register of the column are already empty.

Note also that the constant 3 can be diminished to any $c > 2$ if several registers are used to code "questions". □

As we have seen, the constant C can be effectively found. The member $2n$ cannot be diminished because for every input register we need at least two instruction.

An analogon of Theorem 9.1 can be proved also for some variants of register machines. For example, for the modified register machines from 2.2.2 the expression in Theorem 9.1 can be replaced by $C + n + 2\lceil\sqrt{n}\rceil$ (and the member n cannot be diminished).

10. Some concluding remarks

The author intends to consider some of the ideas below in his future work; some other are presented rather as suggestions for (thinking about and) a discussion.

1. Lower bounds. To obtain a lower bound for the number of instructions of universal R-machines we can use the following statement (it concern the basic variant of R-machines):

Theorem 10.1. *If every strongly connected component of the flowchart of a register machine M contains at most 8 instructions then the halting problem for M is decidable.*

For the equivalence problem of two R-machines the exact bound in a similar statement is known: the undecidability begins with components of cardinality 8; see [7]. So we have the lower bound 9 for the number of instructions of universal R-machines (this bound belong to the upper bound 29; we obviously ought to consider both bounds for the same class of machines). The attempts to increase the lower bound very soon

meet some unsolved number theoretical problems. For example, a connection with the $3x + 1$ problems arises by 14 instructions. (We do not claim the equivalence, only a relationship.)

2. Speed of simulation. The machine U_{32} simulates every R3-machine M in linear time, with the proportionality constant C_M depending on M. The linear time is usually considered as very good. However, in our case the constants C_M (dependent on the numbers of simulated machines) are so huge that the computations cannot be performed in any reasonable time (with exception of some very trivial cases). Of course, this is not a big surprise.

3. Very strong instructions. Let us allow that the activity of an R-machine in one step depends on (the emptiness or nonemptiness) of all n their registers. (An instruction will have the form (q_j, F) where F is a mappping of $\{0, 1\}^n$ to $\{-1, 0, 1\}^n \times \{q_0, q_1, \ldots\}$; above we have excluded such classes of machines.) Then we can simulate 2^k inner states by k additional registers. Hence there is a universal register machine with one instruction and with two inner states, including the final one. The answer to our main question is trivial in this case.

4. The notion of "one instruction". The trivial answer in the previous paragraph was caused by too strong "instructions". Maybe, a more adequate notion of "one instruction" must be found for this case. A natural suggestion for the class of R-machines considered there seems to be that an instruction is an ordered $(2n+2)$tuple of the form $(q_j, a_1, \ldots, a_n, q_k, b_1, \ldots, b_n)$ where $a_i \in \{0, 1\}$ represents the actual states of registers (empty/non-empty) and $b_i \in \{-1, 0\,1\}$ represents operations with them. (Then a machine will be a finite set of such instructions which does not contain any two distinct instructions which coincide in the first $n + 1$ components.) So one original instruction is divided into 2^n new instructions; maybe, some of them are unnecessary. The trivial solution above stops to be valid. However, such approach too strongly influences also the counting of instructions for R-machines considered, e.g., in the Main Theorem. Maybe a "wildcard" for both 0, 1 could be allowed as a value of a_i; the numbers of instructions will be changed, but not too much (only the nontrivial test instructions will be counted twice).

Another point of view is to restrict the activity prescribed in one instruction. Also for the instructions considered in Section 2 we can ask whether they are sufficiently elementary. How elementary the instruction of R-machines ought to be? Is $\langle \overline{\mathrm{Ri\,ZM}} \rangle$ more suitable as one instruction or ought it be divided into two instructions? If we prefer the second possibility then we could ask something similar also for Turing machines. Three activities, writing a symbol, moving the head and changing the inner state ought to be separated. (This can be done indeed but more tape symbols are necessary.)

Of course, these questions have not the unique right answers. We present them only as a further argument that some decisions must be clearly declared before evaluating the numerical results in the theorems similar to presented above.

5. A simplification for teaching. All hard number theoretical results (i.e. those about distribution of primes) can be omitted if the register R_6 becomes empty every time

when the instruction reader ends its activity. To do that, a cycle consisting of two instructions must be inserted into the instruction reader of U_{32} or U_{29}; for $U_{22}, \ldots, U_{19}$ one new instruction suffices.

6. Variants of simulation blocks. Two new simulation blocks are suggested in Fig. 9. The right-hand one is very small; it use only 4 natural instructions; "natural" means that the corrosponding set of instruction codes $\boxed{Ri := 0}$, $\boxed{Ri\,P}$, $\langle Ri = Rk \rangle$ belongs to widely used ones.

The left-hand block can be used instead of the original simulation block of the machine U_{32}; the only disadvantage is that the instructions concerning R_0 cannot be glued together (what was done when further machines for the Main Theorem were constructedd). However, the block is more symmetric, and probably more suitable for teaching.

The number 9 instructions of the simulation block cannot be diminished if we want to simulate three registers (directly and) completely; remember once more that it concerns the basic variabt of R-machines. However, if we would be able to diminish the number of input arrows from 3 to 2 then we can spare 2 instructions in the decoding block.

7. One-state linear operator algorithms. Every such algorithm is determined by its modulus m and by $2m$ integers a_i, b_i. As an input, it obtains $x \in \mathbb{N}$, and in every step it at first determines $i := x \bmod m$, and depending on i it replaces x by $a_i \cdot (x \operatorname{DIV} m) + b_i$; this procedure halts when the result would be negative. (So we can arrange halting by negative a_i.) Kaščák [6] constructed a universal one-state linear operational algorithm with the modulus 396. Non-trivial cases exist already for the modulus 2 (e.g. concerning the $3x + 1$ problem), hence the gap between proved decidability and proved undecidability is very large. Since, the model is very simple the author suggests to study it (as he also suggested in the past).

8. Perspectives of reductions. Roughly speaking, the instruction reader and the decoder together simulate the finite control unit of any simulated machine, and the simulation block simulates its registers. Maybe, such division into two parts will not be

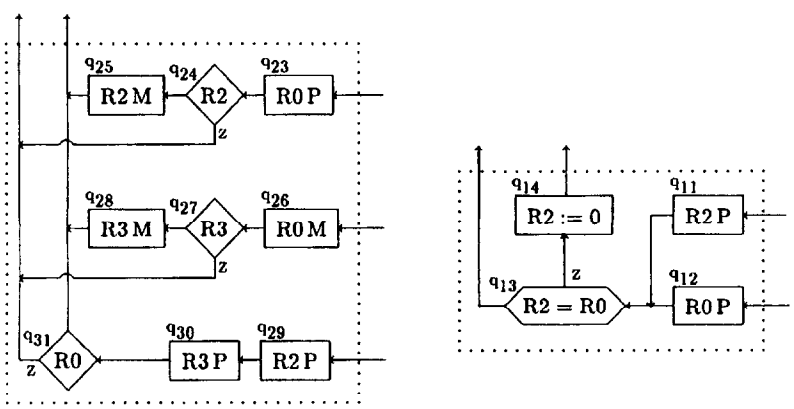

Fig. 9. New suggestion of simulation blocks.

possible for the smallest universal machines. However, while it remains possible the author expects more progress in reductions of the union of the first two blocks.

Acknowledgements

This work was supported by Grant 1224/94 of Slovak Academy of Sciences.

The author acknowledges Prof. M. Margenstern for possibility to present the results at the MCU/UMC'95 Conference, and also many participants (including him) for fruitful discussions. The author also thanks Dr. T. Žáčik for his help with the diagrams of register machines.

References

[1] J.G. Brookshear, *Formal Languages, Automata, and Complexity* (Benjamin/Cummings, Redwood City, 1989).

[2] D.E. Cohen, *Computability and Logic* (Ellis Horwood, Chichester, 1987).

[3] H. Davenport, *Multiplicative Number Theory* (Markham, Chicago, 1967) (Russian translation Moskva, Nauka, 1971).

[4] Ľ. Gregušová and I. Korec, Small universal Minsky machines, in: J. Bečvář ed., Mathematical Foundations of Computer Science, *Proc 8th Symp.*, Olomouc, September 3–7, 1979, Springer Lecture Notes in Computer Science, Vol. 74 (Springer, Berlin, 1979) 308–316.

[5] H. Iwaniec and M. Jutila, Primes in shorts interval, *Ark. Mat.* **17** (1979) 167–176.

[6] F. Kaščák, Small universal one-state linear operator algorithm, in: I.M. Havel, V. Koubek, Eds., Mathematical Foundations of Computer Science, *Proc 17th Symp.*, Prague, August 21–28, 1992, Springer Lecture Notes in Computer Science, Vol. 629 (Springer, Berlin, 1992) 327–335.

[7] I. Korec, A complexity valuation of the partial recursive function following the expectation of the length of their computations on Minsky machines, *Acta F. R. N. Univ. Comen.-Mathematica* **23** (1969) 53–112.

[8] I. Korec, Computational complexity based on expectation of the time of computation on Minsky Machines, in: I.M. Havel, ed., Mathematical Foundations of Computer Science, *Proc. High Tatras*, September 3–8, (1973), VVS OSN, Bratislava, 247–250.

[9] I. Korec, *Introduction to Theory of Algorithms* (In Slovak), PF UK Bratislava, 1974.

[10] I. Korec, Decidability (undecidability) of equivalence of Minsky machines with components consisting of at most seven (eight) elements, in: J. Gruska, ed., Mathematical Foundations of Computer Science, *Proc. 6th Symp.*, Tatranská Lomnica, September 5–9, 1977, Springer Lecture Notes in Computer Science, Vol. 53 (Springer, Berlin, 1977) 324–332.

[11] A.I. Malcev, *Algorithms and Recursive Functions* (in Russian) (Nauka, Moscow, 1965).

[12] M. Margenstern, Nonerasing Turing machines: a frontier between a decidable halting problem and universality, *Theoret. Comput. Sci.* **129** (1994) 419–424.

[13] M.L. Minsky, *Computations: Finite and Infinite Machines* (Prentice-Hall, Englewood Cliffs NJ).

[14] C.J. Mozzochi, On the difference between consecutive primes, *J. Number Theory* **24** (1986) 181–187.

[15] W. Narkiewicz, *Number Theory* (in Polish) (PWN, Warszawa, 1990).

[16] H. Rogers, *Theory of Recursive Functions and Effective Computability* (McGraw-Hill, New York, 1967).

[17] Yu. V. Rogozhin, Seven universal Turing machines (in Russian), *Matematiceskije issledovanija* **69** (1982) 76–90.

[18] K. Weihrauch, *Computability* (Springer, Berlin, 1987).

[19] N.J. Cutland, *Computability* (Cambridge University Press, 1980).