A UNIVERSAL TURING MACHINE WITH 21 INSTRUCTIONS

MICHAEL SCHROEDER

Abstract. We construct a deterministic universal Turing machine with five states, five tape symbols and 21 defined instructions, and prove that its simulation uses polynomial time and space. It has one head on one bi-infinite tape, starts from finite input on a uniform blank background, and halts at an undefined transition. The construction modifies the 22-instruction machine of Neary and Woods: a shorter ordinary-symbol printing token, an adjusted delimiter transition and a compatible encoding preserve their parity-based restoration while eliminating one instruction. We give an explicit compiler from ordinary Turing machines, including unbounded growth at both simulated tape boundaries, and prove positive finite simulation and halting equivalence. For each fixed source machine, the bounds are polynomial in its running time and initial represented tape length. A Lean 4 development verifies the literal transition table, the complete reduction and these bounds. A supplement contains exact costs, output recovery and size optimisation.

1. Introduction

How many instructions suffice for an ordinary universal Turing machine? Turing's universal-machine construction [21] and Shannon's size question [20] lead to two related measures: the numbers of states and symbols, and the number of defined entries in the transition table. We address the latter measure.

The route through tag systems, developed by Minsky [11] and Cocke and Minsky [4], led to Rogozhin's small machines [19] and further state–symbol improvements [8]. Rogozhin's four-state, six-symbol machine has 22 instructions. Neary and Woods matched that count with a five-state, five-symbol machine [14]. We modify their construction to use 21.

Date: 25 September 2026; version 1.0.

Key words and phrases. universal Turing machine, instruction count, bi-tag systems, formal verification.

ORCID: 0009-0004-3249-0195.

DOI: 10.5281/zenodo.22957167.

Theorem 1.1 (Universality with 21 instructions)**.** *The machine U_{21} in Table 2 has exactly 21 defined instructions. There is an explicit computable map $(T, c_0) \mapsto \mathcal{E}(T, c_0)$ from finite ordinary Turing machines and finite initial configurations to finite initial configurations of U_{21} such that*

$$T \text{ halts from } c_0 \iff U_{21} \text{ halts from } \mathcal{E}(T, c_0).$$

Every finite source execution has a finite target execution to a representation of the resulting source state and the complete tape relative to its head. A positive number of source steps is represented by a positive number of target steps. The compiler requires neither a running-time bound nor a bound on future tape use.

The compiler is an explicit finite construction at each level:

$$\text{ordinary TM} \longrightarrow \text{circular rewriting} \longrightarrow \text{bi-tag system} \longrightarrow U_{21}. \tag{1.1}$$

The clockwise and bi-tag frameworks follow Neary and Woods. Earlier circular Post machines also use a growing circular tape [7]. Here a fence and movable marker implement both tape boundaries without a supplied bound on future space.

1.1. **One computation through the encoding.** Consider a source machine with states q_0, q_1, symbols $0, 1$ and blank 0. From q_0 it writes 1, moves right and enters q_1 on either symbol; q_1 has no defined transitions. Starting with the two-cell window $\underline{0}\,0$, its sole step is

$$(q_0, \underline{0}\,0) \longrightarrow (q_1, 1\,\underline{0}) \quad \text{(halt)}.$$

The underline marks the head. Figure 1 follows the input through (1.1). Here F is a fence, M a movable marker, and $0, 1$ abbreviate $\mathsf{Plain}(0), \mathsf{Plain}(1)$.

Circular rewriting. Read the ring from the simulated head, with F separating the right and left portions of the represented window. Removing its first record and appending the written symbol gives

$$(\mathsf{Run}(q_0),\ 0\ 0\ F) \longrightarrow (\mathsf{Run}(q_1),\ 0\ F\ 1).$$

The first 0 is now the scanned cell; the 1 beyond F lies to its left.

Bi-tag rewriting. Number the records $(0, 1, F, M)$ as (a_1, a_2, a_3, a_4). Each source state has five live controls, ordered

$$\mathsf{Run}, \quad \mathsf{Carry}(0), \quad \mathsf{Carry}(1), \quad \mathsf{Carry}(F), \quad \mathsf{Seek}.$$

Thus $\mathsf{Run}(q_0) = e_1$, $\mathsf{Run}(q_1) = e_6$, and e_{11} is the halt control. The rule $e_1 a_1 \mapsto a_2 e_6$ performs the circular rewrite; three single-record copies then bring the control back to the front:

$$\begin{aligned} e_1 a_1 a_1 a_3 &\longrightarrow a_1 a_3 a_2 e_6 \longrightarrow a_3 a_2 e_6 a_1 \\ &\longrightarrow a_2 e_6 a_1 a_3 \longrightarrow e_6 a_1 a_3 a_2. \end{aligned} \tag{1.2}$$

The rotation is one source of simulation overhead.

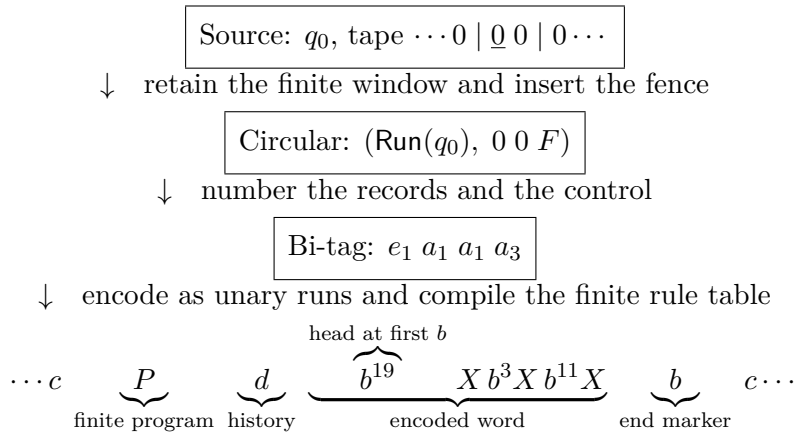

FIGURE 1. One input through the four representations. The physical machine starts in state A. The initial control and first record merge into the run $b^{19} = b^{16}b^3$; X separates runs. The fence F is part of the encoded data. The surrounding c-cells supply unbounded workspace.

The physical tape. Here $q = 4$ record types and $h = 11$ control labels, including halt. The codes defined in Section 3 give $f(a_i) = b^{4i-1}X$ and $f(e_j) = b^{16j}$ for $j < 11$. Each arrow in (1.2) is implemented by one index–print–restore pass. The four unary addresses are $19, 3, 11, 7$. A pass addressed by n adds $n + 1$ history cells, so the history length grows from 1 to 45. At the resulting checkpoint the tape is

$$P\, d^{45}\, b^{99} X\, b^{11} X\, b^{7} X\, b, \qquad 99 = 16 \cdot 6 + 3,$$

with state A and head on the first b after the history. This encodes the successor word and restores the program.

Halting also takes work. The undefined source transition appends $M, 0$ and enters the halt control. After one paired rule and four copies, the word is $e_{11}a_3a_2a_4a_1$. The terminal hardware pass then halts at (E, c). The release checks the source step, both circular endpoints, the bi-tag words and both displayed data encodings in Lean. Its independent literal interpreter checks all ten complete hardware checkpoints. The first source step takes 93,116 transitions; physical halting takes 4,739,378 in total. The finite program has 14,467 cells. Section 7 gives the larger example that also exercises both growing boundaries.

1.2. **Relation to earlier constructions.** The table, backward addressing, local printer and parity-based restoration are inherited from Neary and Woods [14, Section 3.3, Tables 8–11]. The instruction saving couples a three-cell ordinary-symbol printing token with a changed restoration transition and encoding. Their alternating-state parity test is retained with reversed

program and exit parities; it is not a new principle. Section 2 lists all three changed table entries.

TABLE 1. Specified comparators in the ordinary finite-input model. Only defined instructions are counted.

Construction	States	Symbols	Instructions
Rogozhin [19]	4	6	22
Neary–Woods [14]	5	5	22
Present construction	5	5	21

Table 1 establishes the comparison with these published machines, not historical priority against every construction. The 22-instruction benchmark predates Rogozhin's 1996 account and is still reported in the 2015 tribute [24]. We prove an upper bound of 21, with no claim that 20 is impossible.

Model distinctions matter. The 17-rule semi-weak machine of Woods and Neary and their smaller weak machines use repeated infinite backgrounds [23, 15]. Watanabe's displayed 18-instruction construction also uses an infinitely repeated dictionary [27, p. 589, Section 3(b); pp. 590–591, Section 4]. His 21-instruction $(7, 3)$ machine is a separate entry in his Table 1, attributed to a December 1966 MCB report; its construction is not given there. The later survey calls its semi-weak classification an inference [16, Section 3.1, footnote 3]. Its precise background remains unverified here. Nozaki's contemporary discussion also permits periodic *dynamic stops* for Watanabe's simulations [28, p. 34]; this does not establish the physical halting convention used here.

Margenstern and Pavlotskaya [10] prove universality with five instructions and decidability with at most four for a TM connected to a finite automaton. That instruction-count result includes additional machinery outside the counted TM component, so it concerns a different model.

The main paper gives the complete ordinary-machine reduction and its time bound, followed by a computation crossing both source boundaries. The supplement develops exact costs, a sharpness witness, packed encodings, output recovery and size optimisation. Its refinements are not premises of the universality theorem.

2. The machine and its semantics

A source machine has finite nonempty sets Q of states and Γ of symbols, a designated blank $\square \in \Gamma$, and a partial function

$$\delta_T : Q \times \Gamma \rightharpoonup \Gamma \times \{L, R\} \times Q.$$

A configuration is (q, p, t), where $p \in \mathbb{Z}$ and $t : \mathbb{Z} \to \Gamma$. A defined transition changes only $t(p)$, moves the head by -1 or $+1$, and changes the state. A configuration halts when its current pair is outside the domain of δ_T. Every

initial tape considered here equals □ outside a finite interval. Such an input has a finite list representation: list the cells between each supplied endpoint and the head. The library lemma `finite_input_coverage` proves equality with the integer tape, so the formal input type imposes no further restriction. Compilation forgets the absolute coordinate origin; simulation and decoded output retain the entire tape relative to the source head.

Pavlotskaya's decidability results for the two-state, two-symbol and three-state, two-symbol classes [17, 18] assume a reserved halting case, as the survey emphasises [16, Section 1]. Our instruction count excludes undefined halting pairs, consistently with the 22-instruction comparators; see also the explicit counting convention in [9, p. 31]. Reserving a halting case and counting it as an executed instruction are distinct conventions.

The target has states A, B, C, D, E, alphabet g, b, X, c, d, and blank c. In Table 2, bLA means "write b, move left, enter A". A dash denotes an undefined pair.

TABLE 2. The complete transition table of U_{21}.

Read	A	B	C	D	E
g	bLA	gRA	bLC	-	-
b	gLA	gRB	dRE	gRD	dRC
X	cRB	cRB	XRE	cRD	dRA
c	XLA	bLC	XLC	XLC	-
d	bLA	-	bLE	gRB	bLD

The rows have $3, 5, 5, 4, 4$ defined entries, respectively, giving 21. The four undefined pairs are $(D, g), (E, g), (E, c), (B, d)$. There is no separate executed halt instruction or additional halt state. In the published Neary–Woods table, identify $u_1, \ldots, u_5$ with $A, \ldots, E$ and its symbol δ with X. Exactly three entries then differ:

(2.1)

pair	Neary–Woods	U_{21}
(B, d)	gRB	undefined
(D, d)	bLB	gRB
(C, X)	XRC	XRE.

In particular, the relationship to the published 22-instruction machine is not a single-entry modification.

The printer change explains the deleted entry. In the original machine, printing b from the token $dddd$ uses (B, d) after the left move at (D, d) [14, Section 3.3, Cycle 2]. The new (D, d) instruction starts the rightward sweep directly from the token ddd. The proof in Section 4 shows that the modified encoding also avoids (B, d) during indexing and remains compatible with printing and restoration. Changing the parity test alone would not establish this instruction saving.

We write $u \longrightarrow_U u'$ for one defined target step, $u \longrightarrow_U^+ u'$ for a positive finite sequence, and $u \longrightarrow_U^* u'$ for a possibly empty finite sequence. All tape identities below include the state, head position and uniform blank exterior. Finite words display the occupied working interval; explicit leading blank cells may be retained as part of its description. When program size is reported, it is the length of that designated program interval, including its leading header blank.

3. A finite program for a bi-tag system

3.1. **The source rewriting model.** Fix $q \geq 1$ ordinary symbols $a_1, \ldots, a_q$ and $h-1 \geq 1$ live controls $e_1, \ldots, e_{h-1}$, with a distinguished halt control e_h. A valid word has exactly one control and at least one ordinary symbol. Its evolution is determined by

$$a_i z \;\longmapsto\; z a_i, \tag{3.1}$$

$$e_j a_i z \;\longmapsto\; z v_{j,i} \qquad (j < h), \tag{3.2}$$

where each table entry $v_{j,i}$ is either $a_k e_m$ or $a_v a_k e_m$, with $1 \leq m \leq h$. A leading e_h halts. The table contains a rule for every live pair (j, i).

A copy step rotates one symbol. A paired step removes its leading control and record and appends one or two records and a control. Consequently validity is preserved, and the ordinary-record count either stays constant or increases by one. This is the bi-tag family used in [14]. Section 5 gives the explicit source reduction needed here.

3.2. **Unary addresses.** Define

$$\alpha_i = 4i - 1, \qquad \beta_j = 4jq \ (j < h), \qquad \eta = 4hq + 3, \tag{3.3}$$

and extend the following code multiplicatively to words:

$$f(a_i) = b^{\alpha_i} X, \qquad f(e_j) = b^{\beta_j} \ (j < h), \qquad f(e_h) = b^{\eta} X. \tag{3.4}$$

The data word is $f(w)b$, with one final b as sentinel. Live controls have no trailing delimiter: a leading pair $e_j a_i$ therefore has code $b^{\beta_j + \alpha_i} X$. Both kinds of nonhalting address are odd.

There are hq program slots. In increasing address order, let

$$v_{i-1} = a_i \quad (1 \leq i \leq q), \qquad v_{jq+i-1} = v_{j,i} \quad (1 \leq j < h).$$

The first row supplies the copy rules. The remaining rows supply the paired productions, whether their target control is live or halting.

3.3. **The backwards printer code.** Every output v_k begins with an ordinary symbol, whose code begins with at least three b's. Write

$$f(v_k)b = bbu_k. \tag{3.5}$$

For a letter $x \in \{b, X\}$, put

$$\tau(b) = ddd, \qquad \tau(X) = Xdd.$$

If $u = x_1 \cdots x_t$, its printer body is

$$R(u) = \tau(x_t) \cdots \tau(x_1).$$

Thus the first letter to be printed occupies the rightmost three-cell token. If r_k is the number of X's in u_k, then $1 \le r_k \le 3$. Set

$$S(v_k) = (Xd)^{3-r_k} X R(u_k), \qquad P = cd\, S(v_{hq-1}) \cdots S(v_0) X d X d X. \tag{3.6}$$

The X immediately preceding $R(u_k)$ is the stopping delimiter. Any Xd padding lies to its left and is never printed.

Lemma 3.1 (Addresses and parity)**.** *Every slot in* (3.6) *contains four X's, and P contains η of them. The rightmost d of slot k is immediately to the left of the $(4k+3)$rd X counted from the right of P. Every maximal d-run in P is positive and odd, and P has no adjacent X's.*

Proof. A slot contains $3 - r_k$ padding delimiters, one stopping delimiter and r_k body delimiters. The three delimiters in the suffix account for the offset in the address formula. Since slots are stored in reverse address order, each lower slot contributes four more delimiters to the right of the selected one.

For parity, consider the runs in printing order. A first ordinary block gives $3(\alpha_i - 2) + 2 = 12i - 7$ d's; a later ordinary block gives $3\alpha_i + 2 = 12i - 1$. A live control tail together with the sentinel gives $3(\beta_j + 1)$ d's. A halt block gives $3\eta + 2$, followed by a three-d sentinel token. These are all positive odd numbers. The header, padding and suffix runs have length one. Slot boundaries separate, rather than merge, their d-runs. □

3.4. **The invariant configuration.** For positive odd H, define $K(B, w, H)$ to be the configuration

$$\cdots c\, P\, d^H\, \underset{A}{\underline{f(w)}}\, b\, c \cdots, \tag{3.7}$$

where the underline places the head on the first cell of $f(w)$. The program P depends only on B. The compiler takes $H = 1$. Equivalently, one may absorb that first d into $\Pi = Pd$ and write the history as $(dd)^r$, where $H = 2r + 1$.

The history is not reserved workspace. It is the part of the data already consumed by the interpreter. As the data boundary moves right, the history grows; new printed data use the blank tape to the right. No part of the initial configuration encodes a future workspace limit.

4. Printing, restoration and one complete step

4.1. **The two printing tokens.**

Lemma 4.1 (Local printing)**.** *Let v be any finite word over $\{b, X\}$, followed by blank tape. Starting in C on the rightmost d of the displayed token, U_{21} performs the transformations*

$$ddd\, v\, c \;\longmapsto\; bbb\, v\, b, \qquad Xdd\, v\, c \;\longmapsto\; Xbb\, v\, X.$$

In either case it finishes in C on the cell immediately to the left of the token, without reading that cell. The duration is $2|v| + 9$. Every cell outside the displayed interval is unchanged.

Proof. The first two moves are $C/d \mapsto bLE$ and $E/d \mapsto bLD$. For ddd, the third move is $D/d \mapsto gRB$. State B then moves right, marking b as g and X as c, until it writes a new b at the first blank and turns left in C. State C restores the marked corridor while returning. For Xdd, the third move is $D/X \mapsto cRD$; the D sweep similarly reaches the first blank, writes X, and returns in C. The two already written b's and the marked token boundary are restored in each case.

The outward sweep and return each cross the $|v|$ corridor cells. The remaining token and turnaround moves contribute nine steps. Every original corridor delimiter is read as c on the return, because it was marked on the outward sweep. In particular, no returning printer encounters an unmarked X in state C. □

Reading the tokens of $R(u)$ from right to left and applying Lemma 4.1 prints u in its original order. Only d's in the body $R(u)$ become b's. The stopping delimiter and any padding to its left are still unmarked when printing ends. This distinction matters: the entire selected slot need not have been traversed.

4.2. **Selecting the correct slot.**

Lemma 4.2 (Indexing and dispatch)**.** *Suppose the data begin $b^n X z b$, where $n = 4k + 3$ selects slot k. From* (3.7)*, the machine reaches state C on that slot's rightmost d. During this execution it appends one b to the right of the old sentinel. All traversed program and consumed input runs have become b-runs, and the traversed delimiters have been restored to X. The unvisited program prefix is unchanged.*

Proof. After j input b's have been processed, the head is in A at the next input cell. Between that cell and the jth indexed program delimiter, the processed ordinary cells are marked g and the indexed delimiters are marked c. Everything strictly to the left of that delimiter remains unchanged. The case $j = 0$ has no marked program region.

On the next b, state A sweeps left, restoring g to b and c to X, and changing newly reached d's to b. It stops at the next unmarked X, marks that X as c and enters B to the right. The B sweep marks the traversed b's and X's as g and c, respectively. It reaches the marked input boundary in state B on g and returns to A one cell farther right. This proves the invariant by induction.

After n rounds, A reads the input delimiter. Its A/X instruction initiates a final B sweep through z and the old sentinel. At the first blank it appends b and returns in C. The marked corridor is restored, and the return reaches the first untouched d immediately to the left of the nth program delimiter.

Lemma 3.1 identifies that cell as the selected body's rightmost d. Every newly traversed d was changed to b before a B sweep crossed it. Thus the deleted (B, d) instruction is never required. □

The old sentinel and the automatically appended b supply the two initial letters in (3.5). The printer supplies the remaining u_k. If the suffix z ends in a live control, its delimiter-free unary block simply precedes these two new letters; none of that old control block is treated as newly printed output.

4.3. **Adapting the inherited parity test.** Neary and Woods' restoration already alternates between C and E on b [14, Section 3.3, Cycle 3 and Table 11]. Their delimiter instruction $C/X \mapsto XRC$ enters each following run in C: even program runs return to C, whereas the odd exit run reaches E and uses $E/X \mapsto dRA$. Our instruction $C/X \mapsto XRE$ instead enters each run in E. It therefore requires odd program runs and an even exit run. Lemma 3.1 and the history invariant supply precisely these parities. The following lemma proves this adaptation for the literal modified table.

Lemma 4.3 (Parity restoration)**.** *Let $r, m, k_1, \dots, k_r \geq 0$. Starting in C on the first X of*

$$X\,(b^{2k_1+1}X)\cdots(b^{2k_r+1}X)b^{2m}Xv, \tag{4.1}$$

the machine reaches state A on the first cell of v, leaving

$$X\,(d^{2k_1+1}X)\cdots(d^{2k_r+1}X)d^{2m+1}v. \tag{4.2}$$

Every other cell is unchanged. The exact duration is

$$2\left(m+1+\sum_{i=1}^{r}(k_i+1)\right).$$

Proof. The move $C/X \mapsto XRE$ preserves the first delimiter. On b, states E and C alternate while writing d and moving right. An odd run entered in E therefore leaves C at its delimiter. That delimiter is preserved by another C/X move, resetting the state to E for the next run. An even run leaves E at its delimiter; the move $E/X \mapsto dRA$ consumes it and exits. These are all the cases in (4.1). Every move is to the right, so the duration is the number of cells crossed. □

The four instructions used in this lemma are worth displaying:

	b	X
C	dRE	XRE
E	dRC	dRA.

They distinguish the data boundary without storing a program length or testing a special end symbol. For instance,

$$Xb^3Xb^5Xb^4Xv \;\longmapsto\; Xd^3Xd^5Xd^5v$$

takes $2(2+1+2+3) = 16$ steps. The two odd runs retain their delimiters; the even run absorbs its delimiter.

Proposition 4.4 (Complete nonhalting step)**.** *For every valid nonhalting bi-tag word w, every positive odd H, and the source step $w \mapsto w'$, there is a positive finite execution*

$$K(B, w, H) \longrightarrow_U^+ K(B, w', H + n + 1), \tag{4.3}$$

where $n = \alpha_i$ for a copy step and $n = \beta_j + \alpha_i$ for a paired step. No intermediate configuration halts.

Proof. The address formulas select the correct slot by Lemmas 3.1 and 4.2. Lemma 4.1 then prints exactly the output specified by that slot. Printing finishes in C at its stopping delimiter.

Every traversed program run is now an odd b-run. After the final program X, the old history and consumed input form one b-run of length $H+n$, which is even. Lemma 4.3 therefore preserves every program delimiter, restores every visited program run, and consumes exactly the input delimiter. The resulting history has length $H + n + 1$, again positive and odd. The head is in A at the next data cell. Unvisited program cells were untouched, and all visited cells have been accounted for. This gives the complete configuration in (4.3).

Each phase is a finite sweep over a finite word, and every used instruction is defined. The first indexing round already makes the execution positive. The argument applies equally to outputs with two ordinary symbols and to outputs targeting e_h. □

4.4. **The final halt.** Write $P^{(b)}$ for P with all d's replaced by b's.

Proposition 4.5 (Exact halting configuration)**.** *For $w = e_h z$, the execution from $K(B, w, H)$ reaches*

$$P^{(b)} b^H f(w) bb = P^{(b)} b^{H+\eta} X f(z) bb, \tag{4.4}$$

with state E and the head on the leading header c of $P^{(b)}$. Its next transition is undefined.

Proof. The address η equals the number of program delimiters. The indexing routine consequently marks all of them. The automatic output and returning C sweep then reach the remaining header d. The instruction $C/d \mapsto bLE$ puts E on the header blank. There is no (E, c) instruction. This final sweep changes every program d to b, restores all delimiters, and appends the second terminal b shown in (4.4). □

Lemma 4.6 (Positive simulations reflect halting)**.** *Let a relation between two deterministic transition systems map each defined source step to a positive finite target execution, and map each related source halt to a finite target halt. Then, from related configurations, the source halts if and only if the target halts.*

Proof. Preservation follows by concatenating the finite simulations of a halting source run and then its terminal halt routine. For reflection, induct on

the number m of target steps to a halt. If the source already halts, there is nothing to prove. Otherwise its next step has a target simulation of length $\ell > 0$. Determinism and the existence of all ℓ transitions force $\ell \le m$: a longer defined run would cross the purported halt. The remaining target execution has length $m - \ell < m$, and its initial configuration represents the next source configuration. Apply the induction hypothesis. □

Propositions 4.4 and 4.5, followed by Lemma 4.6, prove halting equivalence for every valid bi-tag input. Positivity rules out an infinite sequence of source steps hidden in finitely many target steps; the finite phase proofs rule out an infinite internal sweep.

5. An explicit ordinary-machine compiler

The preceding interpreter is useful only if its inputs can represent unbounded ordinary computations. We now give that reduction directly. Let T have s states and g symbols. Its represented source interval is finite, contains the head and every nonblank cell, and expands by a blank whenever the head crosses either endpoint.

5.1. **One fence and one marker.** A circular configuration consists of a finite nonempty list of records and a control. The first record is current. A rewrite replaces that record by a word v of length one or two and moves to its old successor:

$$(e, a_1 a_2 \cdots a_N) \longmapsto (e', a_2 \cdots a_N v). \tag{5.1}$$

This convention specifies the move even when the replacement has length two. It is convenient to view the list cyclically, starting at its current record.

Use $g + 2$ records

$$\mathsf{Plain}(a) \quad (a \in \Gamma), \qquad F, \qquad M.$$

The fence F marks the cut between the right and left ends of the represented interval. The marker M temporarily stands for the current source cell. The live controls are

$$\mathsf{Run}(q), \quad \mathsf{Carry}(q, r), \quad \mathsf{Seek}(q), \qquad q \in Q, \quad r \in \{\mathsf{Plain}(a) : a \in \Gamma\} \cup \{F\}.$$

There are $s(g + 3)$ of them, and one additional halt control. For readability, plain records will often be written just as a.

There are three clean representations of a source configuration whose current symbol is a:

(i) $\mathsf{Run}(q)$ on $\mathsf{Plain}(a)$, with no marker;
(ii) $\mathsf{Carry}(q, \mathsf{Plain}(a))$ on M, interpreting M as a;
(iii) $\mathsf{Seek}(q)$ on M, interpreting M as a fresh blank.

Each clean ring has one fence. In the last two cases it also has one marker, precisely at the current source cell. Starting just after the fence gives the source interval in left-to-right order. The third representation is used only when the current source symbol is blank.

5.2. **The complete rewrite table.** Define a dispatch function $D_T(q, a)$ with value (replacement, control):

$$D_T(q,a) = \begin{cases} (\mathsf{Plain}(b), \mathsf{Run}(q')), & \delta_T(q,a) = (b, R, q'), \\ (M, \mathsf{Carry}(q', \mathsf{Plain}(b))), & \delta_T(q,a) = (b, L, q'), \\ (M\mathsf{Plain}(a), \mathsf{Halt}), & \delta_T(q,a) \text{ undefined.} \end{cases} \tag{5.2}$$

The remaining rules are in Table 3. Every row uses the old-successor convention (5.1).

TABLE 3. The finite circular compiler. Here r and x are nonmarker records. A dispatch entry means the pair in (5.2).

Control and current record	Replacement	Next control
$\mathsf{Run}(q), \mathsf{Plain}(a)$	$D_T(q,a)$	
$\mathsf{Run}(q), F$	MF	$\mathsf{Seek}(q)$
$\mathsf{Run}(q), M$	M	$\mathsf{Run}(q)$
$\mathsf{Carry}(q,r), x$	r	$\mathsf{Carry}(q,x)$
$\mathsf{Carry}(q, \mathsf{Plain}(a)), M$	$D_T(q,a)$	
$\mathsf{Carry}(q,F), M$	FM	$\mathsf{Seek}(q)$
$\mathsf{Seek}(q), M$	$D_T(q, \square)$	
$\mathsf{Seek}(q), x$	x	$\mathsf{Seek}(q)$

The $\mathsf{Run}(q), M$ case completes the finite table but is not used at a reachable clean configuration. A separate error control is unnecessary. Temporary sweep configurations need not have exactly one fence: carrying the fence can briefly remove its physical occurrence, and a boundary replacement can briefly introduce a second occurrence. The invariant is restored at each clean endpoint. Imposing a one-fence condition at every intermediate rewrite would be incorrect.

Lemma 5.1 (Right motion and its frontier)**.** *A defined right source move has a positive finite circular execution to the correct next clean configuration. It allocates one blank record exactly when the source head crosses the represented right endpoint.*

Proof. Dispatch writes $\mathsf{Plain}(b)$ and advances in $\mathsf{Run}(q')$. If the old successor is a plain record, it is exactly the next source cell. If it is F, the source move crosses the right endpoint. The rule $F \mapsto MF$ puts a fresh marker immediately before the fence and enters $\mathsf{Seek}(q')$. A finite clockwise pass leaves every nonmarker record unchanged and reaches M. Interpreting that marker as blank gives the newly appended rightmost cell and the correct head position. □

Lemma 5.2 (Left motion and its frontier)**.** *A defined left source move has a positive finite circular execution to the correct next clean configuration. It allocates one blank record exactly when the source head crosses the represented left endpoint.*

Proof. Dispatch leaves M at the old source head and carries the written symbol $\mathsf{Plain}(b)$ clockwise. On each subsequent nonmarker record x, the rule writes the carried record and carries x instead. If the old clockwise list after the head is $x_1 \cdots x_k$, the finite carry pass returns to M with carried record x_k and ring

$$M\,\mathsf{Plain}(b)\,x_1 \cdots x_{k-1}.$$

Thus every other old record has shifted one place clockwise, and M represents the old predecessor of the source head.

If $x_k = \mathsf{Plain}(a)$, the control $\mathsf{Carry}(q', \mathsf{Plain}(a))$ on M is a clean representation of the new source head. Cutting at F shows that the source write and left move are exact. The next source transition can be dispatched directly from this representation.

If $x_k = F$, the old head was the leftmost represented cell. The rule $M \mapsto FM$ reinstalls the fence and inserts a fresh marker immediately after it. A finite seek pass reaches that marker, which now represents a blank preceding the old leftmost cell. The carried write b is immediately to its right, as required. □

Proposition 5.3 (Ordinary-to-circular simulation)**.** *The compiler above preserves complete source configurations at clean endpoints. Every defined source step has a positive finite circular simulation. An undefined source transition reaches* Halt*, with M immediately preceding the actual source-head record.*

Proof. All three clean modes dispatch the same source transition: the current symbol is either read from a plain record, stored in the carry control, or known to be blank. Apply Lemmas 5.1 and 5.2. Every carry and seek pass terminates on the unique marker in a finite ring. The undefined dispatch replaces the current record by $M\mathsf{Plain}(a)$ and halts, preserving the current source symbol and its position relative to every other source record. No other dispatch from a clean endpoint halts. □

5.3. **From circular rewriting to bi-tag rewriting.** Fix source enumerations $q_0, \ldots, q_{s-1}$ and $\gamma_0, \ldots, \gamma_{g-1}$. The compiler numbers the records $a_1, \ldots, a_{g+2}$ in the order

$$\mathsf{Plain}(\gamma_0), \ldots, \mathsf{Plain}(\gamma_{g-1}), F, M.$$

For each $i = 0, \ldots, s-1$, in that order, the next block of $g+3$ live controls is

$$\mathsf{Run}(q_i),\ \mathsf{Carry}(q_i, \mathsf{Plain}(\gamma_0)), \ldots, \mathsf{Carry}(q_i, \mathsf{Plain}(\gamma_{g-1})),$$
$$\mathsf{Carry}(q_i, F),\ \mathsf{Seek}(q_i).$$

Thus the block starts at $e_{i(g+3)+1}$, and Halt is $e_{s(g+3)+1}$. This is the order in the Lean definitions `recordAlphabet` and `liveAlphabet`; all fixed-four numerical examples below use it. Each circular rule $(e, a) \mapsto (v, e')$ becomes the paired bi-tag rule $ea \mapsto ve'$. Starting at $ea_1 \cdots a_N$, one paired step gives

$$a_2 \cdots a_N v e'.$$

Rotate the $N-1+|v|$ ordinary records preceding e' by copy steps. The result is $e'a_2 \cdots a_N v$, exactly the circular successor. The same rotations bring a newly produced halt control to the front. The intermediate words remain valid, and each circular rewrite takes a positive finite number of bi-tag steps.

Proof of Theorem 1.1. Represent the finite source interval by a clean ring, enumerate the finite record and control alphabets, construct the complete paired table, and form $K(B, w, 1)$. These are finite operations on the supplied table and input; none executes the source. Translate the target tape so that its initial head is at coordinate zero in state A. The initial nonblank support is finite.

Proposition 5.3, the circular-to-bi-tag reduction, and Proposition 4.4 compose to preserve every finite source execution, with positivity for every defined source step. Their representation relations account for the complete tape and state. The halt routines compose as well, so Lemma 4.6 gives both directions of halting equivalence. The instruction count follows directly from Table 2. □

6. End-to-end simulation time

The three translations in (1.1) have different costs. A left source move can require a full carry pass around the ring; each circular rewrite then requires a full bi-tag rotation. Finally, the hardware traverses a growing history region. Keeping the ring size separate from the number of rewrites is essential to the bound.

Fix the source machine T. Let $\ell \geq 1$ be the number of cells in its initial represented interval, including the head cell and any explicit blanks. Count defined transitions only. The simulation time starts with the finite compiled input already on the tape; it does not include the execution of an external compiler. Constants subscripted by T may depend on its finite transition table and the fixed encoding, but not on ℓ or the source running time.

Proposition 6.1 (End-to-end time and space)**.** *Suppose T executes $t \geq 0$ defined steps, and put $R = \ell + t + 2$. The fixed-four compiler simulates this execution in*

$$S = O_T\Big((tR^2 + R)^2\Big) \tag{6.1}$$

defined transitions of U_{21}, using a tape interval of length $O_T(tR^2 + R)$. If the source halts after these t steps, the same bounds include the complete target halt routine. In particular, for $\ell \leq t + 1$ the time is $O_T((t+1)^6)$ and the space is $O_T((t+1)^3)$.

Proof. Each defined source step allocates at most one cell. Thus every clean source interval has at most $\ell + t$ cells. The circular representation adds one fence. An undefined dispatch inserts one further record to preserve the final head position. Carry and seek rewrites do not otherwise increase the record count, so all intermediate rings and bi-tag words contain at most R ordinary records.

For an interval of m source cells, the simulations in Lemmas 5.1 and 5.2 have the following exact circular costs:

Source move	Circular rewrites
Right, within the interval	1
Right, across its endpoint	$m + 2$
Left, within the interval	$m + 1$
Left, across its endpoint	$2m + 3$

The right frontier uses one dispatch, one insertion and m seek rewrites. A left move uses one dispatch and m carry rewrites; at the frontier it also uses one insertion and $m + 1$ seek rewrites. These counts apply in all three clean modes. The next dispatch acts directly on a carried or fresh marker, so there is no additional pass between source steps. Since $m \le \ell + t$, each defined step costs at most $2R$ circular rewrites. One further dispatch suffices for a source halt. Hence the total number C of circular rewrites is bounded by

$$C \le 2tR + 1. \tag{6.2}$$

If a circular rewrite leaves N' records, its bi-tag simulation consists of one paired rewrite and exactly N' copy rewrites. Since $N' \le R$, the total number M of bi-tag rewrites satisfies

$$M \le (2tR + 1)(R + 1). \tag{6.3}$$

This also includes the rotations that bring a newly produced halt control to the front.

For the fixed compiled bi-tag program B_T, the program length, halt address η, production code lengths and printing-token counts are constants. Let H_i be the history length after i bi-tag rewrites. The fixed-four addresses satisfy $3 \le n_i \le \eta - 4$, and

$$H_0 = 1, \qquad H_{i+1} = H_i + n_i + 1, \qquad 1 + 4i \le H_i \le 1 + (\eta - 3)i. \tag{6.4}$$

The unconsumed data suffix has length $O_T(R)$, since it contains at most R records and one control, each with bounded code length. The exact formulas in Supplement, Section S1 therefore bound the ith hardware macrostep by $K_T(i+R+1)$ for some constant K_T. The same bound applies to the terminal halt routine at $i = M$. Increasing the constant when necessary gives

$$S \le K_T \sum_{i=0}^{M} (i + R + 1) \le K_T (M + 1)(M + R + 1). \tag{6.5}$$

Equations (6.3) and (6.5) imply (6.1), including the case $t = 0$.

The program occupies a constant interval, the history has length $O_T(M+1)$, and the encoded data has length $O_T(R)$. Each macrostep extends the right endpoint by at most the bounded length of its printed production; its sweeps stay between the program header and that endpoint. Thus the whole visited interval has length $O_T(M+R) = O_T(tR^2+R)$. Substituting $\ell \le t+1$ proves the last assertion. □

6.1. **Comparison and scope.** With an initial interval of $O(t+1)$ cells, the successive rewrite counts are quadratic in t for the circular machine, cubic for the bi-tag system, and sixth-power for the hardware. Neary and Woods state the same $O(t^6)$ time bound for their small bi-tag simulators [14, Section 1]; their Theorem 2.1 gives the cubic source-to-bi-tag bound. Thus the 22-to-21 instruction reduction preserves that stated exponent. Constant factors and individual-input times are outside this comparison.

Other small-machine constructions have better published time bounds. Woods and Neary's efficient 2-tag simulation [22] gave polynomial simulation for the classical tag interpreters; Neary's improvement [12, Corollary 5.1.2] gives $O(t^4 \log^2 t)$ for those machines, including Rogozhin's. The larger direct simulators in [13] achieve $O(t^2)$ time. Thus the present result saves an instruction relative to the specified 22-instruction machines and preserves the bi-tag construction's stated exponent; it does not match the faster bound available for Rogozhin's 22-instruction construction. The bounds concern fixed source machines and the respective encodings, rather than a common constant-cost benchmark.

The explicit ℓ in (6.1) is necessary: a quickly halting source can have an arbitrarily large input that the target must traverse. A bound solely in t would not cover this uniformly. For positive t and $\ell = O(t+1)$, the bound gives the conventional $O(t^6)$ statement. The source transition table remains fixed.

The always-left source attains $\Theta(t^6)$ under the prescribed checkpoints; see Supplement, Section S2. The sharpness claim concerns this encoding and simulation schedule.

7. A computation crossing both tape boundaries

Let the source alphabet be $\{0,1\}$ with blank 0, and take states q_0, q_1, q_2, q_3. For either scanned symbol a, define

$$\delta(q_0,a) = (1,R,q_1), \quad \delta(q_1,a) = (0,L,q_2), \quad \delta(q_2,a) = (1,L,q_3),$$

with all transitions from q_3 undefined. Start at coordinate zero on an all-blank tape, represented initially by one cell.

The first step encounters F from Run and allocates a blank before the fence. The third encounters M while carrying F and allocates a blank after the fence. The same fence distinguishes the two boundaries because the controls differ. The compiled bi-tag system has four records and 20 live

TABLE 4. Three source steps and their clean circular representations. The circular list starts at the current record. In a seek mode, M represents blank; in a carry mode, it represents the carried symbol.

Step	Head	Source interval	Circular control	Ring
0	0	$[\underline{0}]$	$\mathsf{Run}(q_0)$	$0F$
1	1	$[1, \underline{0}]$	$\mathsf{Seek}(q_1)$	$MF1$
2	0	$[\underline{1}, 0]$	$\mathsf{Carry}(q_2, 1)$	$M0F$
3	-1	$[\underline{0}, 1, 0]$	$\mathsf{Seek}(q_3)$	$M10F$

controls. Lean proves the three source steps, the resulting source halt and the compiled hardware halt.

Using the numbering in Section 5.3, the fixed-four program has 48,715 cells and the initial tape description has 48,749. Literal execution gives the cumulative counts in Table 5. The source halt dispatch adds one circular rewrite; its rotations put the halt control at the front after 61 bi-tag rewrites. The target then executes its terminal routine in a further 14,533,868 transitions, halting at (E, c). The final count is therefore 374,784,102 defined hardware transitions.

TABLE 5. Cumulative execution counts for Table 4. The terminal routine adds hardware steps without another bi-tag rewrite.

Checkpoint	Circular	Bi-tag	Hardware
Source step 1	3	11	9,471,346
Source step 2	6	23	46,089,542
Source step 3	13	55	308,203,058
Halt control at front	14	61	360,250,234
Hardware halt	14	61	374,784,102

The companion compares the complete initial tape with a fresh Lean compiler export, then checks the entire tape, head and state after each of the 62 hardware macrosteps, including the terminal routine. Separate Lean certificates verify the alphabet order and the circular and bi-tag endpoints. The literal interpreter executes every transition; the duration formulas only specify when to compare checkpoints.

8. Formal verification and scope

The companion uses Lean 4.32.2 [5], its bundled standard library and no external packages. Formal universal machines and resource analysis precede this work in Matita [1, 2], Isabelle/HOL [25, 26], Coq [6] and Lean [3]. The object verified here is this literal table and its finite compiler, including

ordinary integer-indexed source and target tapes. No universality theorem for an intermediate model is assumed.

The principal declaration is

`U21.universal_with_21_instructions`.

It conjoins the exact instruction count with halting equivalence for every finite source table and input. The declaration `simulates_finite_execution` in `Ordinary.Machine` gives positive finite simulation. The independently defined integer-tape steps write only the scanned cell, move one cell left or right, and halt exactly at undefined pairs. The finite representations expand at both ends. The integrated interface theorem `universal_finite_integer_input` in `U21.Referee` covers every integer tape with finite blank exterior.

The quantitative theorems in `Ordinary.Machine` are

`standard_time_space_bound`, `standard_time_space_halt`.

They quantify over both initial length and source running time. Put

$$D = t(\ell + t + 2)^2 + \ell + t + 2.$$

An explicit positive constant K_T, computed from the fixed source table, bounds time by $K_T D^2$ and space by $K_T D$. Space bounds include every intermediate hardware configuration. The halt theorem includes the complete terminal routine. These quantified proofs are separate from finite tests or experimental curve fitting.

The submission verifier builds a fresh extraction of the pinned Lean release. It checks the complete transition table, audits all exported declarations transitively, and rejects deliberately inserted postulates, unfinished proofs and native-evaluation proof axioms. Only `propext`, `Classical.choice` and `Quot.sound` occur; the instruction-count theorem has no axiom dependencies. The compiler definitions are executable. Classical choice in a proof witness does not supply a source-dependent oracle to the compiler.

The boundary-crossing execution in Table 5 is a finite corroboration: its complete input agrees with a fresh Lean export, and a separate literal interpreter checks all 62 full configurations. Its exact numerical runtime is not asserted as a new Lean theorem. The general simulation and complexity results are kernel-checked independently. The supplement records the remaining interfaces and checks, and the distribution gives exact reproduction commands.

The consolidated proof release is

`U21_Lean_v1.0.zip`,

with SHA-256

`b6745fb61109630b3b6d81ee47f119a4`
`06b399ce5735a1cc27bbe67564953a70`.

The five general input-interface lemmas are now in the library module `U21.RefereeBridge`. The walkthrough and the frontier and time-analysis certificates are also library modules. A single invocation of `python3verify.py`

builds all 72 modules, checks the cited interfaces, audits dependencies and runs the executable checks. The submission-level verifier also executes the manuscript examples and rebuilds both documents; `VERIFICATION.json` records its clean run. Kernel checking certifies the stated propositions; human review of definitions, historical priority and the prose remains necessary. The DOI on the title page is reserved for an unpublished Zenodo draft for this article and its companion materials. No public deposit or journal submission is claimed.

Acknowledgments

The construction, formalization and preparation of this manuscript benefited from research assistance by AI systems developed by OpenAI. Responsibility for the mathematical claims, implementation and presentation rests with the author. The debt to Neary and Woods' machine and simulation framework is explicit throughout the paper.

References

[1] A. Asperti and W. Ricciotti, *Formalizing Turing machines*, in Logic, Language, Information and Computation—WoLLIC 2012, Lecture Notes in Comput. Sci. **7456**, Springer, 2012, 1–25. doi:10.1007/978-3-642-32621-9_1.

[2] A. Asperti and W. Ricciotti, *A formalization of multi-tape Turing machines*, Theoret. Comput. Sci. **603** (2015), 23–42. doi:10.1016/j.tcs.2015.07.013.

[3] M. Carneiro, *Formalizing computability theory via partial recursive functions*, in Interactive Theorem Proving—ITP 2019, Leibniz Int. Proc. Inform. **141**, Schloss Dagstuhl, 2019, 12:1–12:17. doi:10.4230/LIPIcs.ITP.2019.12.

[4] J. Cocke and M. Minsky, *Universality of tag systems with $P = 2$*, J. ACM **11** (1964), no. 1, 15–20. doi:10.1145/321203.321206.

[5] L. de Moura and S. Ullrich, *The Lean 4 theorem prover and programming language*, in Automated Deduction—CADE 28, Lecture Notes in Comput. Sci. **12699**, Springer, 2021, 625–635. doi:10.1007/978-3-030-79876-5_37.

[6] Y. Forster, F. Kunze and M. Wuttke, *Verified programming of Turing machines in Coq*, in Certified Programs and Proofs—CPP 2020, ACM, 2020, 114–128. doi:10.1145/3372885.3373816.

[7] M. Kudlek and Y. Rogozhin, *Small universal circular Post machines*, Comput. Sci. J. Moldova **9** (2001), no. 1, 34–52.

[8] M. Kudlek and Y. Rogozhin, *A universal Turing machine with 3 states and 9 symbols*, in Developments in Language Theory—DLT 2001, Lecture Notes in Comput. Sci. **2295**, Springer, 2002, 311–318. doi:10.1007/3-540-46011-X_27.

[9] M. Margenstern, *Turing machines with two letters and two states*, Complex Systems **19** (2010), no. 1, 29–43. doi:10.25088/ComplexSystems.19.1.29.

[10] M. Margenstern and L. Pavlotskaya, *On the optimal number of instructions for universal Turing machines connected with a finite automaton*, Int. J. Algebra Comput. **13** (2003), no. 2, 133–202. doi:10.1142/S0218196703001262.

[11] M. L. Minsky, *Size and structure of universal Turing machines using tag systems*, in Recursive Function Theory, Proc. Sympos. Pure Math. **5**, Amer. Math. Soc., 1962, 229–238. doi:10.1090/pspum/005/0142452.

[12] T. Neary, *Small universal Turing machines*, Ph.D. thesis, National University of Ireland, Maynooth, 2008.

[13] T. Neary and D. Woods, *Small fast universal Turing machines*, Theoret. Comput. Sci. **362** (2006), nos. 1–3, 171–195. doi:10.1016/j.tcs.2006.06.002.
[14] T. Neary and D. Woods, *Four small universal Turing machines*, Fund. Inform. **91** (2009), no. 1, 123–144. doi:10.3233/FI-2009-0036.
[15] T. Neary and D. Woods, *Small weakly universal Turing machines*, in Fundamentals of Computation Theory—FCT 2009, Lecture Notes in Comput. Sci. **5699**, Springer, 2009, 262–273. doi:10.1007/978-3-642-03409-1_24.
[16] T. Neary and D. Woods, *The complexity of small universal Turing machines: a survey*, in SOFSEM 2012: Theory and Practice of Computer Science, Lecture Notes in Comput. Sci. **7147**, Springer, 2012, 385–405. doi:10.1007/978-3-642-27660-6_32.
[17] L. M. Pavlotskaya, *Solvability of the halting problem for certain classes of Turing machines*, Math. Notes **13** (1973), no. 6, 537–541; translated from Mat. Zametki **13** (1973), no. 6, 899–909. doi:10.1007/BF01163965.
[18] L. Pavlotskaya, *Dostatochnye uslovija razreshimosti problemy ostanovki dlja mashin T'juring* [Sufficient conditions for the halting problem decidability of Turing machines], in Avtomaty i Mashiny, 1978, 91–118 (in Russian).
[19] Y. Rogozhin, *Small universal Turing machines*, Theoret. Comput. Sci. **168** (1996), no. 2, 215–240. doi:10.1016/S0304-3975(96)00077-1.
[20] C. E. Shannon, *A universal Turing machine with two internal states*, in Automata Studies (C. E. Shannon and J. McCarthy, eds.), Ann. of Math. Stud. **34**, Princeton Univ. Press, 1956, 157–165. Reprint: doi:10.1515/9781400882618-007.
[21] A. M. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proc. London Math. Soc. (2) **42** (1937), 230–265. doi:10.1112/plms/s2-42.1.230.
[22] D. Woods and T. Neary, *On the time complexity of 2-tag systems and small universal Turing machines*, in Foundations of Computer Science—FOCS 2006, IEEE, 2006, 439–446. doi:10.1109/FOCS.2006.58.
[23] D. Woods and T. Neary, *Small semi-weakly universal Turing machines*, Fund. Inform. **91** (2009), no. 1, 179–195. doi:10.3233/FI-2009-0039.
[24] D. Woods and T. Neary, *Yurii Rogozhin's contributions to the field of small universal Turing machines*, Fund. Inform. **138** (2015), nos. 1–2, 251–258. doi:10.3233/FI-2015-1198.
[25] J. Xu, X. Zhang and C. Urban, *Mechanising Turing machines and computability theory in Isabelle/HOL*, in Interactive Theorem Proving—ITP 2013, Lecture Notes in Comput. Sci. **7998**, Springer, 2013, 147–162. doi:10.1007/978-3-642-39634-2_13.
[26] J. Xu, X. Zhang, C. Urban, S. J. C. Joosten and F. Regensburger, *Universal Turing Machine*, Archive of Formal Proofs, 2019, updated 2022.
[27] S. Watanabe, *4-symbol 5-state universal Turing machine*, Information Processing Society of Japan Magazine **13** (1972), no. 9, 588–592 (in Japanese). IPSJ original scan.
[28] A. Nozaki, *On the notion of universality of Turing machine*, Kybernetika **5** (1969), no. 1, 29–43. DML-CZ original text.

Independent Researcher