# SUPPLEMENT TO A UNIVERSAL TURING MACHINE WITH 21 INSTRUCTIONS

MICHAEL SCHROEDER

Abstract. The 21-instruction machine in the main paper simulates ordinary Turing machines in polynomial time and space. This supplement supplies exact hardware costs, a sixth-power time sharpness witness, a small printer example, packed encodings, output recovery and size optimisation. The full proofs and formal interfaces are retained. References without an S-prefix point to the main paper; S-prefixed references are internal to this supplement.

The universality proof in the main paper uses the fixed-four compiler. The refinements below are proved consequences and alternative encodings of the same hardware. They do not enter the proof of its universality.

## S1. Exact duration of a hardware step

Place the header blank of $P$ at coordinate zero. For a nonhalting canonical configuration write the finite word as

$$Pd^H b^n X z b,$$

and set

$$\begin{aligned} p &= |P| + H, & Z &= |z|, \\ x_1 &> \cdots > x_\eta, & s &= x_n - 1, \\ r_0 &= p + n + 1 + Z, & t &= |u|, \qquad \ell = s - 3t. \end{aligned}$$

Here $x_j$ is the $j$th program delimiter counted from the right, $s$ is the selected body's rightmost $d$, $r_0$ is the old sentinel position, and $bbu$ is the appended output including its new sentinel. The stopping delimiter is at $\ell$.

**Proposition S1.1** (Nonhalting duration)**.** *The complete step in Proposition 4.4 takes exactly*

$$\begin{aligned} T = {} & 2np + n^2 - 2\sum_{j=1}^{n} x_j \\ & + 2(t+1)(p+n+Z-s) + 4t^2 + 12t + 5 \end{aligned} \tag{S1.1}$$

*defined target transitions.*

*Date*: 25 September 2026; version 1.0.
ORCID: 0009-0004-3249-0195.
DOI: 10.5281/zenodo.22957167.

*Proof.* On the $(k+1)$st indexing round, for $0 \le k < n$, the head starts at $p+k$ and reaches delimiter $x_{k+1}$. The outward-and-return cost is $2(p+k-x_{k+1})+1$. Summing gives

$$I = 2np + n^2 - 2\sum_{j=1}^{n} x_j.$$

The automatic output and return have cost $Z+3+r_0-s$. Thereafter the initial printer corridor has length $V_0 = r_0+1-s$. Each printed letter advances the output end by one cell and moves the next token three cells left, so the $k$th corridor has length $V_0+4k$. Lemma 4.1 therefore gives total printing cost

$$\sum_{k=0}^{t-1}\bigl(2(V_0+4k)+9\bigr) = t(2V_0+9)+4t(t-1).$$

Finally, restoration moves right from $\ell$ to $p+n+1$, costing $p+n-\ell+1$. Add these four quantities and substitute the definitions of $r_0, V_0, \ell$ to obtain (S1.1). □

**Proposition S1.2** (Halting duration)**.** *If the source word starts with $e_h$ and its remaining encoding has length $Z$, the halt routine takes exactly*

$$T_{\mathrm{halt}} = 2\eta p + \eta^2 - 2\sum_{j=1}^{\eta} x_j + p + \eta + 2Z + 4 \tag{S1.2}$$

*defined target transitions.*

*Proof.* Index all $\eta$ delimiters using the same round count. The automatic output and return end at the header $d$ in position one. One further $C/d$ move puts the head on the header blank in state $E$. Adding these moves gives (S1.2). The undefined pair itself contributes no executed step. □

Both formulas are translation invariant. For a fixed bi-tag system and initial history, addresses and production lengths are bounded, while history and encoded word lengths after $N$ rewrites are $O(N+|w_0|+1)$. Summation, including a terminal halt routine, gives $O((N+1)(N+|w_0|+1))$ time. Section 6 refines the record-count estimate using the source interval and composes all three simulations.

## S2. A computation attaining the sixth power

The exponent is sharp for this compiler's prescribed checkpoints. This is not a lower bound for the hardware under arbitrary encodings.

**Proposition S2.1** (Sharpness for the fixed-four compiler)**.** *There is a one-state, one-symbol source machine, started on one blank cell, for which the fixed-four simulation through the first $t$ source steps takes $\Theta(t^6)$ hardware transitions as $t \to \infty$.*

*Proof.* Let the only source instruction write blank, move left and retain the same state. Before step $j+1$, the represented interval has $m = j+1$ cells, and the move crosses its left endpoint. It requires $2m+3 = 2j+5$ circular rewrites. The dispatch and carry pass comprise $m+1$ rewrites leaving $m+1$ records each; insertion and seeking comprise $m+2$ rewrites leaving $m+2$ records each. The corresponding bi-tag count is therefore

$$(m+1)(m+2) + (m+2)(m+3) = 2(m+2)^2.$$

Summing over the first $t$ source moves gives exact counts

$$C_t = t^2 + 4t, \qquad M_t = 2\sum_{j=0}^{t-1}(j+3)^2 = \frac{2t^3+15t^2+37t}{3}. \tag{S2.1}$$

For macrostep $i$, indexing alone costs

$$I_i = 2n_i(|P| + H_i) + n_i^2 - 2\sum_{r=1}^{n_i} x_r \;\geq\; 2n_iH_i \;\geq\; 6H_i,$$

because every delimiter coordinate is less than $|P|$ and $n_i \geq 3$. By (6.4), the hardware cost through these $M_t$ rewrites is at least

$$6\sum_{i=0}^{M_t-1}(1+4i) = 12M_t^2 - 6M_t = \Omega(t^6).$$

Proposition 6.1 supplies the matching upper bound with $\ell = 1$. □

The first three endpoints give a concrete check on all three levels. With the numbering in Section 5.3, the four live controls are Run, Carry(Plain(□)), Carry($F$), Seek, in that order:

| Source steps | Circular rewrites | Bi-tag rewrites | Hardware steps |
|---|---|---|---|
| 1 | 5 | 18 | 926,148 |
| 2 | 12 | 50 | 3,210,968 |
| 3 | 21 | 100 | 7,661,040 |

Literal execution gives these hardware counts, with the complete tape, head and state checked at each bi-tag checkpoint. Separate Lean examples verify the compiled table and three circular and bi-tag endpoints. The exact hardware counts depend on this numbering through the encoded addresses; the general asymptotic exponents do not. The general upper bound, fixed-four sharpness, and packed-layout extension are also formalised; Section 8 gives their quantified interfaces and verification scope.

## S3. A growing production, then a halt

Take $q = 1$, $h = 2$, and the single paired rule

$$e_1a_1 \longmapsto a_1a_1e_2. \tag{S3.1}$$

The addresses are $\alpha_1 = 3$, $\beta_1 = 4$, $\eta = 11$. The copy and paired output codes, including their sentinels, are

$$b^3Xb, \qquad b^3Xb^3Xb^{11}Xb.$$

The two printer slots are therefore

$$S(a_1) = (Xd)^2Xd^3Xd^5,$$
$$S(a_1a_1e_2) = Xd^3Xd^{35}Xd^{11}Xd^5.$$

The first has two padding delimiters and the second has none. They have lengths 14 and 58, respectively. The complete program

$$P = cd\,S(a_1a_1e_2)S(a_1)XdXdX$$

has 79 cells and 11 delimiters. Its initial configuration has the 89-cell description $Pd\,b^7Xb$, headed at cell 80 when the header blank is at cell zero.

TABLE S1. The entire execution of (S3.1) from $e_1a_1$. Each duration counts defined hardware transitions. The last row is the halt routine, rather than another bi-tag rewrite.

| Current word | $H$ | Address | Duration | Cumulative |
|---|---|---|---|---|
| $e_1a_1$ | 1 | 7 | 3,002 | 3,002 |
| $a_1a_1e_2$ | 9 | 3 | 430 | 3,432 |
| $a_1e_2a_1$ | 13 | 3 | 486 | 3,918 |
| $e_2a_1a_1$ | 17 | 11 | 1,188 | 5,106 |

For the first step, the seven indexed delimiters have positions

$$78,\ 76,\ 74,\ 68,\ 64,\ 62,\ 60.$$

The selected body's rightmost $d$ is at 59, and $|u| = 19$. Indexing costs 205 transitions. Automatic output and return cost 32; the 19 printing tokens cost 2,679; restoration costs 86. Their sum is 3,002. The restoration sweep crosses the body runs and the remaining program runs, then consumes the data delimiter because $H + n = 1 + 7 = 8$ is even.

After two copy rotations, the source halt reaches the front. The halted word is

$$P^{(b)}b^{28}Xb^3Xb^3Xbb. \tag{S3.2}$$

The even run of length $28 = 17 + 11$ will also identify the program–data boundary for the decoder in Section S5. An independent literal interpreter reproduces every complete configuration in Table S1; the example script compares the entire nonblank tape, head and state after each stated duration.

Changing the target of (S3.1) to $e_1$ gives a nonhalting growth example. The program then has 57 cells and the first paired step takes 1,498 transitions. Each circuit of the control adds one ordinary record. Hence its record

count is unbounded, and the general halting-equivalence theorem implies that its hardware execution does not halt. This is also a direct way to see why no fixed initial data allocation can suffice.

## S4. Packed program encodings

Four delimiters per slot give particularly simple addresses. They are not always needed. A copy output and a one-record output with a live target require only two delimiters, including the stopping delimiter. A two-record output or a halt-targeting output requires four after rounding its delimiter requirement up to an even number. Even widths preserve the parity argument.

Index rows by $j = 0, \dots, h-1$, with $j = 0$ the copy row. Let $\kappa_{j,i}$ be the minimum *half-width* of the slot in column $i$: it is one for a copy or a one-record live output and two otherwise. Choose positive integers

$$G_1, \dots, G_{q-1}, \qquad Z_0, \dots, Z_{h-1}$$

such that $G_i \geq \max_j \kappa_{j,i}$ and $Z_j \geq \kappa_{j,q}$. The half-width of an internal column $i < q$ is $G_i$ in *every* row; the final column has half-width $Z_j$. The common internal gaps ensure that an ordinary address has the same meaning in all rows.

Define

$$A_i = \sum_{t<i} G_t, \qquad L_j = \sum_{t<q} G_t + Z_j,$$

$$O_j = \sum_{r<j} L_r \quad (j \geq 1), \qquad L = \sum_{r=0}^{h-1} L_r. \tag{S4.1}$$

Replace the address formulas by

$$\alpha_i = 2A_i + 3, \qquad \beta_j = 2O_j, \qquad \eta = 2L + 3. \tag{S4.2}$$

Keep the word code (3.4), now using these addresses. For a slot of half-width $W$ and output $bbu$, with $r$ delimiters in $u$, use

$$(Xd)^{2W-r-1} XR(u). \tag{S4.3}$$

The capacity condition makes the exponent nonnegative. Retain the same header, reverse slot order and suffix as in (3.6). Choosing every half-width equal to two recovers the fixed-four code.

**Theorem S4.1** (Packed simulation and minimum budgets)**.** *Every admissible choice of the above half-widths has the same complete simulation and halting properties as the fixed-four encoding. At a fixed record order and control order, the choice*

$$G_i = \max_j \kappa_{j,i}, \qquad Z_j = \kappa_{j,q} \tag{S4.4}$$

*minimises both the literal program length and the complete initial input length among all admissible budgets.*

*Proof.* There are $2L+3$ program delimiters. Before column $i$ in row $j$ there are $2(O_j + A_i)$ slot delimiters, so its address is $2(O_j + A_i) + 3 = \beta_j + \alpha_i$. All ordinary addresses remain odd and live-control codes even. The printer-body runs are still positive odd runs, and padding contributes runs of length one. Lemmas 4.1–4.3 therefore apply unchanged, including the halting address and complete restoration.

The budget (S4.4) is componentwise no larger than any admissible budget. To check that this also minimises physical length, write a printable tail as

$$u = b^{2k+1} X b^{2k_1+1} X \cdots X b^{2k_\ell+1}.$$

It has $\ell$ delimiters. Counting its three-cell tokens and its padding in (S4.3) gives the exact slot length

$$|S| = 6\left(k + \sum_{i=1}^{\ell} k_i\right) + 4\ell + 2 + 4W. \tag{S4.5}$$

At fixed rule shapes, increasing any budget weakly increases every address parameter in this formula and leaves $\ell$ unchanged. It also weakly increases each data-code length. Thus program and input lengths are monotone in the budget, proving both minima. □

## S5. Recovering the program and the output

Output recovery is part of the classical universality specification. Rogozhin discusses its importance and explicitly allows a decoder whose sole argument is the halted configuration [6, Section 2, Remark 3]. The result below supplies a concrete inverse for our entire packed family: it recovers the labelled program and its layout as well as the simulated output.

At a compiled halt, the head is on the header blank in state $E$. The finite observed word starts at that cell and ends just before the next blank to its right. Its nonempty suffix contains only $b$ and $X$, so this scan requires no external length. Explicit exterior blank padding makes no difference to the observation. Denote this word by $\mathrm{obs}(u)$ for a halted configuration $u$.

**Theorem S5.1** (Uniform program inverse)**.** *For every canonical halted bi-tag computation, under any admissible packed budget, the observed word effectively determines the complete labelled bi-tag table, its physical program, the history length and the final bi-tag word. No dimensions, slot widths, source table or history length are supplied to the inverse.*

*Proof.* We describe the inverse in the order in which information becomes available.

*The boundary.* In (4.4), every program $b$-run is odd. The history and leading halt code form a run of length $H + \eta$, which is even. Thus the first even $b$-run terminated by $X$ identifies that boundary. Its preceding prefix is exactly $P^{(b)}$. Replacing its $b$'s by $d$'s recovers $P$; counting its delimiters gives $\eta$. Subtracting $\eta$ from the even run length gives $H$.

*The slots and their widths.* Remove the known suffix $XdXdX$ and read the program from right to left. Each body consists of tokens $ddd$ and $Xdd$, decoded as $b$ and $X$, until its bare stopping delimiter. The output is then obtained by prepending $bb$. After a body, padding consists of one-$d$ runs. A genuine body begins, in this reading order, with at least three $d$'s, because every output ends in a nonempty odd sentinel run. Consequently padding cannot consume the next body. Counting body, stop and padding delimiters recovers each even slot width.

*The alphabets and rows.* The first slots in address order are the copy row. Their decoded outputs have the form $b^{\alpha_i}Xb$, where successive addresses are determined by the recovered widths. No live production continues this pattern: a one-record live production has a final run of length $\beta_j + 1 > 1$, and a two-record or halt-targeting production has at least two delimiters. Hence the length of the initial copy sequence is exactly $q$. Grouping the recovered slots into rows of length $q$ gives $h$ and all row offsets. The first output run and subsequent runs can now be decoded using (S4.2). This reconstructs every rule, including its source row, source column, output records and target control.

*The final data.* Beyond the even run's delimiter, only ordinary records remain: the unique control is already the leading halt symbol. Remove the final $bb$, decode the ordinary blocks using the recovered addresses, and prepend $e_h$. Reconstruction yields exactly (4.4). All scans and table operations are finite. □

The implemented inverse checks exact reconstruction and returns a typed table with proofs of rectangularity and label bounds. It can therefore be executed as a bi-tag system. Acceptance of an arbitrary word does not itself prove that the word is reachable from a compiled input; reachability is supplied by the simulation theorem when the inverse is applied to an actual runtime halt.

S5.1. **Identifying structural records.** The ordinary compiler places $F$ and $M$ after all plain symbols. A smaller layout may change their positions. The reconstructed table still determines their roles. Every halt-producing two-record rule is a dispatch output $M\mathsf{Plain}(a)$. If an ordinary compiled computation halts, such a rule exists. Its first output identifies $M$ independently of its ordinal. The live two-record frontier rules are $MF$ and $FM$. Once $M$ is known, either identifies $F$; these rules are present for every source state. No other live two-record rules occur in the ordinary compiler.

At the final circular halt, remove $M$ and take its former successor as the source head. Cut after $F$ to recover the source interval. To make the blank exterior independent of metadata, the compiler first swaps the declared source blank with ordinal zero. This is an effective involutive relabelling. The remaining plain-symbol order is preserved by the permitted layouts below.

**Theorem S5.2** (Output from an actual hardware halt)**.** *There is an explicit output compiler $\mathcal{E}_{\mathrm{out}}$ with the halting-equivalence property of Theorem 1.1, and a single computable decoder $D$ taking only the observed halted word, with the following property. If an execution from $\mathcal{E}_{\mathrm{out}}(T, c_0)$ halts at $u$, then the source reaches a halted configuration $c'$ and $D(\mathrm{obs}(u))$ is a finite window $W$ such that, for every $x \in \mathbb{Z}$,*

$$\mathrm{tape}_W(x) = \nu_T(\mathrm{tape}_{c'}(\mathrm{head}(c') + x)). \tag{S5.1}$$

*Here $\nu_T$ is the compiler's blank-first symbol renaming and the exterior of $W$ is zero. The result holds after every permitted structural-record placement and every live-control permutation.*

*Proof.* Halting reflection supplies a source halt. Determinism identifies the given hardware halt with the exact halted configuration reached by simulating that source run. Apply Theorem S5.1, identify $M$ and $F$ by their rules, and apply the circular output construction just described. Removing those two structural records leaves the known order of plain-symbol ordinals. Blank normalisation identifies the entire exterior as zero. The representation invariants then give (S5.1) cell by cell. □

This output convention recovers the whole tape relative to its final head. The initial absolute coordinate origin, arbitrary human-readable symbol names and the halted state name are not encoded as output metadata. Nor does the inverse undo the source computation to recover its original input. For any fixed source alphabet with blank ordinal zero, equation (S5.1) is an ordinary uniform output interface.

## S6. Exact size optimisation within the packed family

Fix a record order and use the minimum budgets of Theorem S4.1. Only the live-control row order remains to be chosen. Let $L_e > 0$ be the half-width of live row $e$, and let $c_e$ be the number of paired rules whose target is $e$. Let $i_e = 1$ when $e$ is the initial live control and $i_e = 0$ otherwise. For an initially halted word every $i_e$ is zero. For a permutation $\pi$ of the live rows, put

$$p_e(\pi) = \sum_{d \text{ before } e \text{ in } \pi} L_d.$$

The fixed copy row precedes all live rows in address order.

**Lemma S6.1** (Literal size accounting)**.** *There are constants $C_P, C_I$, independent of $\pi$, such that the actual program and input lengths are*

$$P(\pi) = C_P + \sum_e 6c_e\, p_e(\pi), \tag{S6.1}$$

$$I(\pi) = C_I + \sum_e (6c_e + 2i_e)\, p_e(\pi). \tag{S6.2}$$

*Proof.* The program has seven header-and-suffix cells plus the sum of its slot lengths (S4.5). Reordering live rows changes only the code lengths of their target controls. Each occurrence of a live target $e$ contributes its half-offset with coefficient six to its printer body. There are $c_e$ such occurrences. The copy-row offset, all halt-target offsets, all ordinary addresses, all padding widths and all rule shapes are independent of the live-row permutation and belong to $C_P$.

The input consists of the program, one history cell and the initial encoded data. There is exactly one initial control; if it is live, its code contributes its half-offset with coefficient two. All remaining data contributions are independent of $\pi$. This gives (S6.2). □

The factors six and two therefore come from physical cells: three printer cells per encoded letter and two unary letters per half-offset. They are not surrogate weights introduced as an optimisation heuristic.

**Theorem S6.2** (Optimal live-row order)**.** *For lexicographic minimisation of $(P, I)$, attach to row $e$ the weight pair $w_e = (6c_e, 6c_e+2i_e)$. For minimisation of $(I, P)$, reverse this pair. Order rows so that every earlier row $a$ and later row $b$ satisfy*

$$L_a w_b \leq_{\text{lex}} L_b w_a. \tag{S6.3}$$

*Every such order is globally optimal over all live-control permutations. Zero weights and equal ratios are allowed.*

*Proof.* For two adjacent rows $a, b$, the order $a, b$ contributes $L_a w_b$ to their mutual weighted-prefix cost, while $b, a$ contributes $L_b w_a$. Their common prefix contribution and the contribution to later rows are unchanged. Thus (S6.3) is exactly the condition under which placing $a$ first does not increase the objective pair.

Because all $L_e$ are positive, comparison of the rational pairs $w_e/L_e$ in reverse lexicographic order is total and transitive. It can be performed by the integer cross-products in (S6.3), without division. Starting from any competitor, move the first row of a sorted order to the front. Each adjacent exchange is non-increasing. Repeat on the remaining rows. Induction proves global optimality. The proof uses no positivity assumption on the weights and so includes zero weights and ties. □

For one objective this is Smith's scheduling rule [7]. Replacing weighted completion times by weighted prefix times subtracts the order-independent quantity $\sum_e w_e L_e$. The two-objective version merely uses the secondary comparison when the primary costs agree. The specific contribution here is Lemma S6.1 and its application to this compiler.

For an ordinary source, allow every record order that preserves the plain-symbol chain and keeps its first, blank record first. The fence and marker can occupy any two distinct remaining positions. If there are $q = g + 2$ records, this gives exactly $(q - 1)(q - 2)$ placements. For each, recompute

the minimum budgets, sort its live rows by (S6.3), and compare the resulting literal size pairs.

**Corollary S6.3** (Optimal permitted layout)**.** *The preceding finite procedure returns a global lexicographic minimum among all permitted record placements and all live-control permutations using minimum admissible packed budgets. Its compiled input retains the universality and output properties proved above.*

*Proof.* The placement enumeration is complete: choose the distinct positions of $F$ and $M$ after the first record and fill the remaining positions with the plain-symbol chain. For each such placement, Theorem S6.2 minimises the live-row order. Taking the least of these finitely many minima gives the global minimum. Record and control renamings preserve the circular and bi-tag semantics, while Theorems S4.1 and S5.2 cover the resulting encodings. □

The algorithm replaces factorial control-order enumeration by sorting. The Lean implementation uses merge sort and a proved size formula, so it need not materialise each full candidate tape. The theorem is about the output size and the executable algorithm; it is not a formal wall-clock bound on Lean evaluation.

*Example* S6.4 (Equal frequencies, different row lengths)*.* With one ordinary record and two live controls, take

$$e_1a_1 \mapsto a_1a_1e_2, \qquad e_2a_1 \mapsto a_1e_1,$$

and initial word $e_1a_1$. The row half-widths are $L_1 = 2$ and $L_2 = 1$, while $c_1 = c_2 = 1$. Ordering by frequency alone cannot distinguish them. Condition (S6.3) puts $e_2$ before $e_1$. In the original order $(P, I) = (75, 83)$; in the reversed order $(P, I) = (69, 79)$. The program saving is $6(2 - 1) = 6$ cells. The initial control code grows by two cells, leaving an input saving of four. Both objective modes choose the reversed order. These sizes and their optimality are also kernel-checked examples.

Tape size and execution time are different objectives. An immediately halting one-state, one-symbol source gives the following measured comparison:

| Packed encoding | Program cells | Input cells | Defined steps |
|---|---|---|---|
| Original order | 1,903 | 1,927 | 152,198 |
| Size-optimised order | 1,517 | 1,575 | 300,282 |

Both rows are literal hardware executions with complete final-tape comparison and subsequent decoding. The smaller program takes more steps. No runtime dominance is asserted by Corollary S6.3.

## S7. Formal verification and reproducibility

The companion is written in Lean 4 [4]. The pinned toolchain is version 4.32.2. It uses the bundled standard library and has no external Lean package dependencies. The release contains 72 library modules and 662 written theorem declarations, including 13 modules for quantitative bounds. Source and target each have an independent integer-indexed tape semantics; the finite list representations used in the proofs are related to those semantics by proved refinement theorems.

S7.1. **Prior formal developments.** Asperti and Ricciotti verified universal machines in Matita, first for single tapes and then for multiple tapes [1, 2]. Xu, Zhang and Urban's Isabelle/HOL development [8] is the antecedent of the AFP entry [9]. In Lean, Carneiro formalised a universal partial recursive function and undecidability of halting [3]; that development uses a different computational model.

Forster, Kunze and Wuttke [5] verified polynomial time and constant-factor space overheads for universality and for translation from multiple tapes to one. Their interpreter uses multiple tapes, and the resulting single-tape class is parameterised by the alphabet [5, Section 8]. The contribution here is the verification of a fixed 21-instruction table and its concrete encodings, including the whole-execution resource bounds and the restricted-family optimality statements; no claim to the first formalisation of universality or its complexity is made.

S7.2. **What the final theorem states.** The principal declaration is

`U21.universal_with_21_instructions`.

Suppressing only namespace qualifications in the prose, it combines

$$\texttt{definedPairs.length} = 21$$

with a universal quantification over the source state count, symbol count, machine and finite configuration. The conclusion is halting equivalence under the independent source and target `standardStep` functions, using the explicit compiled input. The stronger finite-execution theorem constructs a target run and a relation to the complete next source configuration, with positivity when the source run has positive length.

The formal chain follows the mathematical structure of this paper: literal tokens and sweeps; odd-run program construction; complete slot dispatch; bi-tag simulation; circular simulation; ordinary-tape refinement; and their composition. Source universality and compiler correctness are proved conclusions, rather than assumptions in the final theorem. Both boundary extensions are included in the ordinary simulation.

Table S2 gives a compact declaration map. Names after the first entry are relative to namespace `U21`. The companion file `CheckInterfaces.lean` gives an executable check of these interfaces.

TABLE S2. Mathematical claims and their Lean interfaces. Long namespace prefixes are split across lines for readability.

| Claim | Principal declaration |
|---|---|
| Count and universality | `U21.universal_with_21_instructions` |
| Finite source execution | `Ordinary.Machine.`<br>`simulates_finite_execution` |
| Finite compiled input | `Ordinary.Machine.compiled_finite` |
| Program restoration | `restoration`, `program_slot_standard` |
| Packed simulation | `Packed.program_slot`<br>`Ordinary.Machine.packed_universal` |
| Full abstract inverse | `Packed.Budget.recover_table` |
| Runtime source output | `Ordinary.Machine.`<br>`layout_runtime_output_correct` |
| Physical cost formula | `Packed.Budget.sizes_accounting` |
| Global row-order optimum | `Layout.sortRows_optimal` |
| Fast layout optimum | `Layout.fastOptimise_spec` |
| Optimised output compiler | `Ordinary.Machine.`<br>`fastOutput_universal`<br>`fastOutput_runtime_correct` |
| External schedule checker | `Layout.certifySchedules_sound` |

S7.3. **Quantitative interfaces.** The time and space theorems construct actual finite executions of the literal target table. For $D = t(\ell + t + 2)^2 + \ell + t + 2$, the library defines a positive, executable constant $K_T$, depending only on the source table, and proves

$$n \le K_T D^2, \qquad \text{visited tape span} \le K_T D.$$

The prefix theorem concludes a representation of the reached source configuration. The halt theorem instead concludes an actual undefined target transition, with its complete halt routine included in $n$. Both have interfaces using the independent integer-tape source semantics. For $\ell \le t + 1$, the explicit bounds are $144K_T(t+1)^6$ transitions and $12K_T(t+1)^3$ cells.

The space predicate quantifies over every intermediate target time, including all printing and restoration sweeps. Its interval contains every head position and every nonblank cell. The sharpness proof connects the exact counts in (S2.1) to those same literal executions, and proves a natural-number $\Theta(t^6)$ relation. The observational runtime function selects a proved execution witness by classical choice; the compiler, phase costs and circular/bi-tag schedules remain executable definitions.

For each fixed packed budget, the quantitative coefficient contains its actual program and code sizes. For minimum budgets under varying orders,

a finite sum over all record/control permutations supplies a single coefficient independent of the input. The resulting ordinary theorem applies directly to the fast optimiser and its blank normalisation. Table S3 identifies these results.

TABLE S3. Quantitative declarations, relative to namespace `U21`.

| Claim | Principal declaration |
|---|---|
| Entire execution | `Runs.bounded` |
| Prefix and halt | `Ordinary.Machine.`<br>`standard_time_space_bound`<br>`standard_time_space_halt` |
| Time and space | `Ordinary.Machine.`<br>`standard_sixth_power_halt` |
| Sharpness | `Complexity.LeftWitness.`<br>`time_theta_sixth` |
| Packed budgets | `Packed.Complexity.time_space_bound` |
| Layout family | `Layout.Choice.time_space_bound` |
| Fast compiler | `Ordinary.Machine.`<br>`fastOutput_sixth_power_halt` |

S7.4. **The verification boundary.** The transitive dependency audit finds only three logical axioms:

`propext`, `Classical.choice`, `Quot.sound`.

The instruction-count theorem has no axiom dependencies. There are no unfinished proofs, additional mathematical postulates or native-evaluation proof axioms. The compiler definitions are executable finite constructions.

Kernel checking establishes the propositions attached to these definitions. The correspondence between those definitions and the intended machine model remains an appropriate subject for human review, which is why both the literal table and the integer-tape interfaces are exposed. Formal verification does not establish historical priority or an instruction-count lower bound. It also does not automatically certify the prose of a manuscript or every external utility in its companion.

The coordinate-form exact-duration identities in Section S1 remain written derivations from the phase counts. The Lean asymptotic proofs use the verified phase counts and affine slot-duration identities directly; they do not assume those coordinate formulas. In contrast, both Proposition 6.1 and Proposition S2.1 have quantified Lean theorems, as do the upper bounds for the packed-layout extension.

S7.5. **Independent finite checks.** The release verifier compares all 25 table entries with a pinned reference, compares complete compiled inputs with

independent Python encoders, and executes selected inputs in an independent literal interpreter. Runtime checks compare every nonblank cell, the head and the state, then feed the observed result to the Lean decoder. Synthetic inverse fixtures are recorded separately from actual executions. The examples in Table S1 have an additional small literal-execution script in the manuscript companion. The time-analysis companion checks 324 finite window cases across all three clean modes, and executes the 100 hardware macrosteps used in Section S2. The growing-left example has an integrated certificate module, `U21.TimeBoundsExamples`. It binds the complete compiled table and the first three source, circular and bi-tag endpoints to the pinned definitions. These finite certificates corroborate the examples; the quantified complexity theorems establish the general bounds independently of those checks.

For Section 7, an additional C interpreter executes all 374,784,102 transitions and checks the 62 complete configurations. Its 25 table entries are compared with the pinned reference. Numbering and intermediate endpoints are certified in `U21.FrontierExamples`. The introductory example of Section 1.1 is verified in `U21.Walkthrough`. Its complete compiled input is also executed independently through ten full hardware checkpoints. All three witness modules are built and audited by the normal release verifier.

The fast-layout suite also certifies 20 concrete outputs of a pinned Python optimiser. A Lean checker enumerates every permitted structural placement itself, checks the submitted control schedules by integer comparisons, and recomputes the sizes. Its soundness theorem proves global optimality whenever the checks accept. The generated finite certificates are checked by kernel reduction. The suite also checks rejection of 20 wrong-role layouts and 18 permitted but inferior layouts. These results certify the submitted outputs; they do not formalise Python language semantics.

The finite checks bind executable components and examples to the mathematical construction. They are not premises of the quantified simulation, inverse or optimisation theorems. In particular, large or long-running examples are not needed to justify unbounded tape growth.

S7.6. **A pinned companion.** The main paper and this supplement are tied to the local release `U21_Lean_v1.0.zip`. Its SHA-256 digest is

```
b6745fb61109630b3b6d81ee47f119a4
06b399ce5735a1cc27bbe67564953a70
```

The line break is typographical; the digest is a single 64-digit hexadecimal string. From the submission directory, run:

```
python3 verify.py
```

The same command checks the `U21.RefereeBridge` module and requires all five general input lemmas. These include finite-input coverage and universality for integer tapes with finite blank exterior. The manuscript interfaces are checked automatically. No supplementary proof file needs to be copied into the project or checked separately. The command checks both manifests,

builds a fresh Lean extraction, executes the manuscript examples and rebuilds both PDFs. `VERIFICATION.json` records the clean run. Generated compiler inputs, certificates and intermediate reports are reproduced during verification; they are not needed in the source archive. The DOI on the title page is reserved for the unpublished Zenodo draft for the article and companion materials.

## 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] 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.

[5] 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.

[6] Y. Rogozhin, *Small universal Turing machines*, Theoret. Comput. Sci. **168** (1996), no. 2, 215–240. doi:10.1016/S0304-3975(96)00077-1.

[7] W. E. Smith, *Various optimizers for single-stage production*, Naval Res. Logist. Quart. **3** (1956), nos. 1–2, 59–66. doi:10.1002/nav.3800030106.

[8] 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.

[9] J. Xu, X. Zhang, C. Urban, S. J. C. Joosten and F. Regensburger, *Universal Turing Machine*, Archive of Formal Proofs, 2019, updated 2022.


Independent Researcher