

TECHNICAL SUPPLEMENT TO A UNIVERSAL TURING MACHINE WITH 20 INSTRUCTIONS

MICHAEL SCHROEDER

ABSTRACT. This supplement identifies the exact interfaces behind the manuscript’s claims, records the physical resource constants and output-reader costs, and describes the frozen formal companion and the checks performed for the paper. It distinguishes quantified Lean proofs, finite literal executions, accepted compiler certificates and research cost calculations.

1. SCOPE AND NOTATION

The main manuscript’s universality and resource theorems concern the reference direct compiler. Its final input policy additionally supports checked candidates and minimum initial-size selection. These are different policies on the same twenty-instruction hardware. The following notation is used here:

Symbol	Meaning
T	Fixed original ordinary source machine
x, n, N	Input word, $ x $, and $\max(1, n)$
t	Defined original source transitions before halt
C	Reference certificate <code>DirectStop.compile</code> T $T.blank$
P, P_{aux}	Active program bank and refresh adapter
λ, d_{ep}	Template length and source epoch period $\lambda + 3$
h	Active history in pairs, hence $2h$ physical cells
q	Physical queue length, including unary fields and tail
w	Weight in the compositional physical protocol estimate
u	Backend-source time after optional frontend transformations

Every displayed bound on U20 time counts ordinary defined transitions of its literal partial table. An undefined transition stops execution; it has no additional transition cost. Work needed to *reach* that pair is charged. The external decoder is a distinct machine, and its transition count must not be silently added as if it used the same single-tape instruction set.

2. THE PUBLIC STATEMENTS AND THEIR QUANTIFIERS

Let c_0 be the independent integer-tape interpretation of source input (T, q_0, x) , and let a be any optional candidate value. The final compiler

constructs a program and matching physical reader

$$R = \text{SizeBoundedCompiler.compile}(T, q_0, x, a).$$

Its public universality/readback theorem asserts all of the following:

- (1) the fixed hardware table has 20 defined pairs;
- (2) there exist finite integer endpoints outside which R 's initial tape is the uniform target blank;
- (3) its represented initial size is at most $I_T(n + 1)$;
- (4) source and target halting are equivalent;
- (5) whenever the independent source semantics runs for t steps to a halted c , there exist $m > 0$ and a literal halted U20 configuration z reached in exactly m steps, and the matching reader physically runs from z to a representation of c .

The Lean name is

`U20.Finalized.finite_input_universality_and_readback.`

The source convention is explicit: state q_0 , head zero, input symbols at $0, \dots, n - 1$, and the designated source blank everywhere else. Empty input scans that blank. All finite tables, start states, words (including words containing blanks) and candidate values are quantified. Nonempty source sets follow from the designated blank and start state, each represented by an element of the corresponding finite set. The reference target starts in state A at head zero. Neither reference compilation nor its reader table takes t , a future space bound or a halting certificate as an argument.

The resource theorem uses the *same interface with* $a = \text{none}$. It adds the reference time bound and an every-prefix physical interval bound. Its name is

`U20.Finalized.reference_polynomial_physical.`

The proof first obtains a reference resource witness and a physical readback witness. The lemmas `Exec.terminal_time_unique` and `Exec.terminal_unique` identify their times and final configurations. Thus the resource and output assertions concern one execution, rather than two potentially unrelated existential witnesses.

2.1. What physical output means. The output predicate is defined by existence of a finite reader run:

$$\begin{aligned} \text{run}(\text{readerStep}, D, \text{readerInitial}(z)) &= \text{some}(f), \\ \text{readerStep}(f) &= \text{none}, \quad \text{observe}(f) = \text{some}(c). \end{aligned}$$

It also bounds reader workspace at every prefix by $16 + 5D$. The reader initialization specifies its physical tape roles. Its input is the actual target tape, and its observation maps the output tapes to an original-source configuration. Exactness is equality of the state, absolute head and function $\mathbb{Z} \rightarrow \Gamma$, not just equality of a finite printed list. The observation is an interpretation of physical output tapes containing marked left/right words and finite

metadata. Computing an integer head coordinate from the origin mark is part of that interpretation. No separately written binary head register or materialized infinite tape is claimed or included in D .

Reader initialization is also a representation boundary. The terminal invariant supplies $z.right = [c]$. Under this invariant, `DirectPhysicalDecoder.halted_input_at` proves that the reader input at coordinate i is the actual target tape at $-i$. The reader starts at the actual halted head, with reversed orientation; it does not copy a saved word or use a mathematical decoder to manufacture its input. In the full frontend, subsequent stages likewise receive the actual preceding output tapes.

2.2. Minimum initial-size selection. Let R_c be the checked candidate result, which may already include fallback, and R_0 the original-source reference result. The compiler returns R_c when $|R_c| \leq |R_0|$ and R_0 otherwise, selecting the physical reader and correctness proof with the program. Consequently

$$|R| = \min(|R_c|, |R_0|).$$

The result is uniform over candidate *values*. It assumes no correctness theorem for a candidate producer, and cannot force a nonterminating producer to supply a value. The branch selected for each of the four historical optimized fixtures has not been evaluated under this new size guard.

3. ASSEMBLER DETAILS AND EXACT LOCAL WORK

3.1. Changes from the preceding U_{21} table. Relative to the preceding U_{21} table, the changes are as follows. This is a coordinated replacement of printing and restoration, not merely deletion of a reachable instruction.

(3.1)	pair	U_{21}	U_{20}
	(C, d)	bLE	bLD
	(C, X)	XRE	XRC
	(E, b)	dRC	gRC
	(E, X)	dRA	gRA
	(E, d)	bLD	undefined
	(D, d)	gRB	undefined
	(D, g)	undefined	gRB

Cleaned program runs have even length, so restoration returns to state C at each program delimiter: $C/X = XRC$ preserves that phase. The cleaned history and consumed odd address have odd total length, so $E/X = gRA$ recognizes the consumed delimiter and closes the last dg pair, consistently with $E/b = gRC$ on ordinary pairs. In the redesigned two-cell printer, $C/d = bLD$ followed by $D/g = gRB$ starts a b -printing sweep directly, while D/X selects an X -printing sweep; live printing therefore needs neither E/d nor D/d , and the undefined D/d is used for the header halt. Parity thus underlies the restoration change, while the coordinated token redesign supplies the shorter printing route. Two entries disappear and one appears.

The net saving is one instruction. The same target table serves both the original bi-tag compiler and the later direct compiler; bounds belong to those particular compilers, not to the table in isolation.

3.2. Slot representation and local costs. The physical frame for parameter $k \geq 0$ is $X(dg)^{k+1}$. It emits $b^k X$. A slot with r frames has an even delimiter width $2v \geq r + 1$; the remaining delimiters include a stop and padding. In the nearest-first Lean representation its block parameters are

$$(k_1 + 1, \dots, k_r + 1, 0) 0^{2v-r-1}.$$

A parameter a contributes a run of $2a$ non-delimiters and one separator. Reversal recovers the main paper’s physical left-to-right slot expression. This convention accounts for both the leading g skip of a frame and the common appended b of a call.

Let V_j be the sum of half-widths preceding a selected slot, μ the frame-output length, h the history pair count, and ℓ_{suf} the remaining data length. The exact local formula is

$$(3.2) \quad T(h, \ell_{\text{suf}}) = T(0, 0) + 4(2V_j + \mu + 2)h + (2 + 2\mu)\ell_{\text{suf}}.$$

It comes from exact indexing, production and restoration recurrences. The corresponding theorem exhibits a literal run of that length. The analogous raw-bank theorem permits explicitly chosen padding and an arbitrary archived prefix; neither is reinitialized at a checkpoint.

The following names identify the complete chain:

Interface	Content
<code>program_slot_standard</code>	Complete assembled live call
<code>program_halt_standard</code>	Standalone header halt
<code>Cost.slotTime_affine</code>	Exact formula (3.2)
<code>Cost.slot_execution</code>	Literal run of the computed cost
<code>Cost.slot_bounded</code>	Every-prefix interval bound
<code>RawProgram.execute</code>	Raw-bank call preserving archives
<code>RawProgram.time_affine</code>	Raw-bank exact affine formula
<code>RawProgram.bounded</code>	Raw-bank physical interval bound

All names in this supplement are under namespace `U20` unless explicitly prefixed otherwise.

3.3. The 51-cell fixture. The three slots are $((), 1)$, $((2), 1)$ and $((2, 9), 2)$, where the second component is the half-width. Their physical sizes are 2, 8, 30. Together with the two-cell header and final delimiter they form a 43-cell bank. The eight-cell data word is $bXbXb^3X$. The initial configuration has state A , head coordinate 43 and total represented size 51.

The queue trace is

$$\begin{aligned} ((1, 1, 3), 0, 0) &\rightarrow ((1, 3), 1, 1) \rightarrow ((3), 2, 2) \\ &\rightarrow ((5), 0, 4) \rightarrow ((3, 9), 0, 7) \rightarrow ((9, 3), 0, 9), \end{aligned}$$

with components queue, unfinished tail and history pairs. The final header dispatch reaches state D on d at coordinate zero. The exact full-run theorem is `ResearchFixture.exact_execution`; `no_premature_halt` excludes an earlier undefined pair. `Examples.macro_cost_total` separately proves that the five exact live-call costs and header cost sum to 2,999. The manuscript verifier also regenerates the bank from the slot formula, interprets every literal transition, checks each checkpoint and compares the complete terminal tape with the Lean fixture.

4. THE SOURCE INVARIANT AND FINITE CONSTRUCTION

4.1. Local source cases. A source mark is a symbol and an origin bit. Write $P(m)$ for a plain record, $H(q, m)$ for a headed one and K, G, J for guards/sentinel. The intermediate alphabet contains $P(m)$, incoming-left $L(q, m)$, outgoing-right $R(q, m)$, frozen $F(q, m)$ and $K_L(q)$.

Let $\text{in}_L(H(q, m)) = q'$ when the source action is a left move to q' , and let it be absent in every other case. The g phase on a previous/current pair (l, c) has priority cases $c = K$ (emit J) and $c = J$ (emit G). In the remaining cases it emits the updated previous record:

Previous record	Output
$l = K$	$K_L(q')$ if $\text{in}_L(c) = q'$; otherwise K
$l = H(q, m)$, undefined action	$F(q, m)$
$l = H(q, m)$, write a' , move left	$P((a', m.\text{origin}))$
$l = H(q, m)$, write a' , move right to q'	$R(q', (a', m.\text{origin}))$
$l = P(m)$	$L(q', m)$ if $\text{in}_L(c) = q'$; otherwise $P(m)$
$l = G$ or J	Same previous row with an unmarked designated hot symbol

Here the *hot symbol* is a designated filler symbol used in the boundary cases. The reference chooses the source blank for that role. The finite concurrent copier supplies the boundary/sentinel arrangement required by these priority cases.

Let $\text{in}_R(l) = q'$ exactly for $l = R(q', m)$. The h phase uses the current intermediate record and that incoming state:

Current record	Output
$K; J$	The same guard
$K_L(q)$	$K, H(q, (\square, 0))$
G	$H(q', (\square, 0)), G$ if $\text{in}_R(l) = q'$; otherwise G
$P(m)$	$H(q', m)$ if $\text{in}_R(l) = q'$; otherwise $P(m)$
$R(q, m)$	$P(m)$
$L(q, m)$ or $F(q, m)$	$H(q, m)$

These are mathematical finite lookup tables, not extra U20 instructions. Each is realized through pulse addition and physical slots. The `GuardedStep` lemmas cover interior and frontier moves; `Guarded.source_halt` proves preservation on undefined source input. The terminal protocol uses the frozen event to stop instead of performing endless identity cycles.

4.2. Synchronizing helper layers. Helper graphs generate the pulses in fixed layers. A helper splits an integer weight into children whose weights sum to the parent weight. All leaves arrive in the designated pulse layer; delay/carrier records prevent early lookup. The formal direct compiler uses explicit root, top, middle and carrier rows. A zero-weight sentinel closes each pass. An extra relay aligns the returning pass. If a logical carry record has empty output, the next physical output is reduced by the known pending offset before concatenation. Positivity and cancellation are proved, so no accidental zero or even ordinary call is introduced.

This synchronization is essential: summing the correct number of pulses at the wrong point in the queue would select a different table entry. The theorems `DirectPhase.forward` and `DirectPhase.returning` specify the entire queue before and after each pass, including the final tail value.

4.3. Why symbolic assembly terminates. The compiler enumerates the finite record alphabets and their required adjacent pairs. Each job has a symbolic key, a delimiter width and a finite list of symbolic or literal output references. Prefix sums of widths determine odd addresses. Output-reference resolution may enlarge unary runs but cannot change these delimiter addresses. A finite registry deduplicates helper nodes. Regions distinguish source rows, front-end adapter keys, aliases and serializer nodes.

A unit template denomination guarantees representability of every finite descriptor. The default serializer uses fanout two and one helper level for that denomination. The proof does not posit the existence of a refresh certificate: `DirectAdapter.certified` constructs one from finite parameters, and `DirectAdapter.unitCertified` supplies concrete parameters for every source. A concrete snapshot terminal is then incorporated by `DirectStop.compile`.

The *effectiveness* of this construction does not assert a formal polynomial-time bound on the host compiler. All constants are finite computable functions of the source; their sharp dependence on source table dimensions is outside the final reference theorem.

5. EXPLICIT REFERENCE RESOURCE CONSTANTS

Write F_T for `RawProgram.bankCells C.program`. The following quantities are exact definitions from the resource modules:

$$(5.1) \quad d_{\text{ep}} = \lambda + 3,$$

$$(5.2) \quad B_T = \text{auxiliaryBudget} + 2\lambda + d_{\text{ep}} + 7,$$

$$(5.3) \quad C_q = \text{recordBudget},$$

$$(5.4) \quad C_e = d_{\text{ep}} \text{ cycleBudget RawQueue.budget}(P) + \text{roundTripCoefficient},$$

$$(5.5) \quad H_0 = 1 + \text{RawQueue.budget}(P),$$

$$(5.6)$$

$$K_{\text{epoch}} = 30C_e(1 + H_0 + C_q + C_e).$$

The field **resourceBase** is B_T , and **queueCoefficient** is C_q . The field **epochCoefficient** is C_e , and **resetHistory** is H_0 . The budgets are finite sums/maxima over the constructed finite tables, with all helper expansion calls charged. The exact expressions are in **RefreshSourceCost**, **RefreshLinearCost**, **RefreshEpochCost** and **RefreshRunCost**.

Let $A = \text{terminalCoefficient}$, and let P_{stop} be the concrete snapshot bank. Put

$$(5.7) \quad Z = H_0 + C_q + A,$$

$$(5.8) \quad V = (Z + 2) \text{ RawQueue.budget}(P_{\text{stop}}),$$

$$(5.9) \quad K_{\text{term}} = 30A(1 + H_0 + C_q + A) + (V + \text{haltBudget})(2 + Z + V),$$

$$(5.10) \quad J_{\text{term}} = A + V + 2.$$

The coefficients used in the final theorem are exactly

$$(5.11) \quad K_T = K_{\text{epoch}} + K_{\text{term}}, \quad J_T = C_e + J_{\text{term}}.$$

The terminal halt budget is the computed **PlainBank.haltBudget** of the snapshot values. These quantities include terminal formatting, whose cost cannot be inferred from live slot costs alone.

For $W = B_T + N + t$, the complete theorem gives a run of length $m \leq (t + 1)K_TW^2$ with initial head zero and interval endpoints

$$lo = -F_T, \quad hi = \|Q_0\| + (t + 1)J_TW.$$

The precise predicate requires

$$lo \leq \text{head} \leq hi, \quad z < lo \text{ or } hi \leq z \implies \text{tape}(z) = c$$

at every transition prefix. Nonblank support uses a half-open interval; head positions use its closed extension, allowing the frontier blank. Thus $hi - lo + 1$ safely counts a containing set of cells.

Initial size obeys

$$|\mathcal{E}_T(x)| \leq F_T + C_q(B_T + N) \leq [F_T + C_q(B_T + 1)](n + 1).$$

The last coefficient is **SizeBoundedCompiler.inputCoefficient**. It depends only on the fixed original source, even when an arbitrary candidate value is supplied to the size-selection policy. The explicit retained-space coefficient is

$$S_T = I_T + 1 + J_T(B_T + 1),$$

which bounds the containing interval by $S_T(t+1)(N+t+1)$ in one paper-facing Lean execution theorem. The main paper also gives a streaming-transducer proof of $O_T(n+1)$ encoding time; this is not an operational cost proof for the Lean evaluator.

5.1. Archives and the compositional potential. The physical effect theorem has two separate accounting devices: active history plus queue length controls local execution cost, whereas fixed left boundary and increasing right boundary control retained physical space. They must not be interchanged. An installation can reset the active history to one pair while leaving an arbitrarily long archive behind it. The archive is included in the physical interval but is not repeatedly traversed by later ordinary calls.

A protected descriptor installation has initial queue

$$(2, \text{descriptor}(P'), 1) R$$

and unfinished tail z . It physically installs P' , returns queue R with tail $z+1$, and resets the active history. The next bootstrap is part of the protocol and cancels the known offset. The formal lemma `RawProtocolCost.install` has weight two, but its input length already includes the full physical descriptor. Thus the bound does not treat writing or traversing a large descriptor as constant-time work.

6. CONCRETE REFERENCE COMPILER EVALUATIONS

The two source tables are defined in `verification/ReferenceBenchmark.lean`. The zero-transition case has one state, one symbol (the blank), no defined action, start state zero and empty input. The writer has states and symbols $\{0, 1\}$, blank zero, and $\delta(0, a) = (1, R, 1)$ for both symbols a ; state one has no defined action. It starts in state zero on 00 and halts after one source transition. Both use the reference compiler’s unit power, binary serializer of depth one, and blank hot symbol.

TABLE 1. Exact sizes of the uncompressed reference descriptions.

Quantity	Zero-transition source	One-step writer
Active-bank entries	2,296	4,878
Adapter-bank entries	8,993	17,447
Terminal-bank entries	8,638	22,485
Active-bank cells F_T	1,202,566,981	5,940,353,221
Adapter-bank cells	414,131,901	1,777,530,747
Terminal-bank cells	149,230,093	1,011,150,459
Template length λ	601,250,868	2,970,096,575
Initial queue cells q_0	7,164,507,372,126	78,749,145,577,630
Complete initial cells $F_T + q_0$	7,165,709,939,107	78,755,085,930,851

The active, adapter and terminal bank descriptions were exported by compiled Lean code; the two exports completed in approximately 69

and 334 seconds on the test host. These are host evaluation times, not U_{20} transition counts. The record dictionaries and phase/record budgets come from `ReferenceMetadata.lean`. The staged active bank is proved definitionally equal to `DirectStop.compile`’s program. The theorem `CompactReferenceChecks.input_exact` identifies the root-mapped queue, template and left bank with `DirectStop.input`. The added theorem `CompactReferenceChecks.template_export_exact` identifies the reference template with the unit-power encoding of the parsed descriptor of that very bank. This follows directly from the compiler constructors; parse success is already guaranteed by the bank-format certificate. Thus the experiments describe that reference input, without allocating its expanded trillions of cells or substituting a compressed loader.

6.1. Compact evaluation of the coefficients. For the unit template, a parsed segment of amount a contributes $a + 1$ letters. The first copier cycle changes an initial word of length $\lambda + 2$ to one of length $2\lambda + 4$, and all later cycles preserve that length. The paper-facing Lean file `CompactReferenceChecks.lean` proves these equalities for arbitrary templates. Hence the exact auxiliary sum used by the existing coefficient definitions is

$$\text{auxiliaryBudget} = \lambda + 2 + (\lambda + 2)(2\lambda + 4), \quad B_T = 2\lambda^2 + 12\lambda + 20.$$

This identity permits evaluation without performing $\lambda + 3$ copier cycles on an expanded template. Finite bank budgets are accumulated from their run-length parameters. All integer arithmetic is exact; Table 2 rounds only the displayed values. The full integers, intermediate budgets, source descriptions and hashes are in `verification/reference-benchmark.json`.

TABLE 2. Reference coefficients, rounded to six significant digits. These are evaluated upper-bound coefficients, not measured runtimes.

Coefficient	Zero-transition source	One-step writer
B_T	7.23005×10^{17}	1.76429×10^{19}
K_T	7.06820×10^{96}	2.85794×10^{105}
J_T	2.65861×10^{48}	5.34597×10^{52}
I_T	2.25260×10^{23}	1.59694×10^{25}
S_T	1.92219×10^{66}	9.43186×10^{71}

Here $I_T = F_T + Q(B_T + 1)$ and $S_T = I_T + 1 + J_T(B_T + 1)$. The latter is the explicit coefficient in the new single execution contract for retained space. These estimates are very conservative; neither K_T nor its value after substituting t is an exact complete execution time. For the zero-transition input the exact size is $F_T + q_0 = 7,165,709,939,107$, whereas the stated linear bound is $I_T = 225,260,229,272,561,484,398,866$ cells at $n = 0$. The ratio is

approximately 3.143586×10^{10} . For the writer, $n = 2$, so the comparable bound is $3I_T$, approximately 6.083184×10^{11} times its actual initial size.

6.2. Literal execution and its limits. The independent Python simulator applies exactly one of the twenty hardware instructions per iteration. Unmodified cells are read from a compact description of the finite input, and a write overlay records all changes. Compact storage is a host implementation choice: it does not change the physical tape size or replace a target operation by a macro. Both inputs were executed for exactly 10^6 literal transitions.

TABLE 3. Observed configurations after 10^6 transitions from head zero.

Quantity	Zero-transition source	One-step writer
State and scanned symbol	(B, b)	(B, b)
Head coordinate	$-30,432$	$-79,706$
Visited head interval	$[-73\,126, 17]$	$[-107\,635, 12]$
Distinct cells written	73,144	107,648
Target halted	no	no
Reader executed	no	no

For each case, the exact ordinary-call formula gives the following *calculated* number of transitions for its first queue call:

$$348,312,994,355,389,072, \quad 8,472,500,016,549,451,652.$$

These calls end in the live state A and precede terminal processing. Their enormous costs make a full literal halt run infeasible within this experiment, including the $t = 0$ source. These prefixes do not measure a complete literal halt time. The separate compressed replay below now completes the actual reference macro traces and decodes their terminal tapes; its target transition counts are calculated from exact local formulas, not measured by literal iteration.

The evaluator is cross-checked against Lean evaluations of 32 small bank-budget cases and 16 copier traces. Its compact tape representation is compared cell by cell with 16 fully expanded small inputs. In those 16 cases, the calculated first-call cost is checked by literal execution, including the complete resulting tape. Finally, this same simulator completes the existing 2,999-transition fixture and matches its whole terminal configuration. These tests check the implementation; the exported large numerical evaluations themselves are not kernel reductions of the full tapes or runs.

6.3. Compressed reference replay. A second simulator executes the actual reference output through terminal readback at the level of queue calls.

The script `verification/reference_macro_replay.py` executes the exported bank parameters without consulting a source transition table or a

predicted sequence of source phases. A queue block is a finite address word repeated a natural number of times. This representation is needed because, for example, seed address 11915 in the zero-source bank emits [11923, 12525]: ordinary single-symbol run lengths would expand into many alternating runs.

For each front block, the simulator looks up the actual production for every address in its word. Empty productions increment the unfinished tail; nonempty ones add that tail to their first address and reset it. Only the block already at the front of the FIFO is batched. If its output is nonempty, its trailing empty productions determine a fixed tail for subsequent repetitions, so the first repetition and the remaining repetitions can be represented separately. No newly appended address is processed ahead of an unconsumed input address. Exact local costs are affine in history and physical queue length; summing the arithmetic progression gives the batch's calculated transition count.

An even control address is never passed through this ordinary-call batching. Installation at address two parses the even descriptor codes actually present in the queue, stopping at the produced odd continuation field, and reconstructs the new bank from them. It also updates head, history, tail and retained archive. Both installed terminal banks match their separately exported parameter lists exactly. At the final even empty entry, the simulator constructs the physical terminal tape as compressed $X(gd)^k$ fields using the actual residual queue and history. The undefined instruction is E/c in both cases.

An independent compressed implementation of `DispatchSnapshot.decode` scans those saved physical fields. The static record dictionary from `ReferenceReplayMetadata.lean` then feeds the source decoder. Neither decoder receives the source answer or the pre-terminal source configuration. They recover state zero, head zero and the all-blank tape for the first input, and state one, head one and the single nonblank cell $0 \mapsto 1$ for the writer. These are functional decoder replays on compressed physical data, not step-by-step executions of the formal five-tape reader machine.

TABLE 4. Completed reference macro executions. Target transition counts are calculated by summing exact local costs along the traces.

Quantity	Zero-transition source	One-step writer
Ordinary queue calls	775,644,398,417	36,039,771,581,614
Compressed batches	669,221	4,356,896
Terminal-bank installations	1	1
Calculated target transitions	2.527899×10^{29}	2.644035×10^{32}
Terminal state/symbol	E/c	E/c
Decoded state/head	(0, 0)	(1, 1)

Full integer counts, head coordinates, recovered outputs and source hashes are recorded in `reference-zero-macro-replay.json` and `reference-writer-macro-replay.json`. The complete terminal tapes, including retained archives, are saved as compressed physical descriptions in the corresponding `reference-*-terminal.json.gz` files. The decoders are run after reloading those files. The macro engine is differential-tested against individual U20 transitions in 48 ordinary-batch cases, nine bank installations and 18 terminal halt/readback cases; every comparison checks the entire finite tape, state and head, not merely the decoded answer. Eighteen malformed-header controls are rejected. These checks are in `verify_macro-replay.py` and `macro-replay-validation.json`.

The same checker executes a 7,320-transition physical bank-refresh component $P \rightarrow Q \rightarrow P$. The installed Q produces P 's descriptor, the second installation restores the original bank, two archived banks remain charged to tape, and bootstrap and subsequent queue execution continue. Five full-tape checkpoints match literal simulation. The initial helper-bank descriptor is supplied in this small fixture; it is not a complete autonomous reference-compiler epoch. In particular, the two reference examples halt before their periods $\lambda + 3 = 601,250,871$ and $2,970,096,578$. Their periodic reference refresh therefore remains covered by the proof, not by these completed reference executions.

6.4. A possible template optimization. The reference constructor chooses a single unit power, which makes λ approximately $F_T/2$ in these examples. The general template machinery also supports other positive power schedules containing a unit. Binary powers could shorten descriptor encodings substantially, but they also change the helper banks and expansion depth. A net improvement requires rebuilding and measuring that compiler; none is claimed for the present reference construction.

7. CHECKED OPTIMIZATIONS, STARTUP AND TRANSFORMED SOURCE TIME

7.1. The preceding bi-tag route. For initial represented source length ℓ , the U_{21} companion bounds time by $O_T(D^2)$ and space by $O_T(D)$, where

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

For $\ell = O(t)$ these are sixth-degree time and cubic space. The older bi-tag compiler on the same twenty-instruction hardware has conservative bounds of the same degrees. The main paper's improved exponent belongs to the direct refresh compiler.

7.2. Restricted banks and compressed startup. Many generic source-record pairs are unreachable from a specified initial configuration. A finite inductive neighborhood certificate can establish a closed overapproximation of the reachable pairs. The restricted assembler keeps the required rows and validates their physical addresses and outputs. This is a finite invariant argument, not simulation up to a chosen horizon.

A compressed loader contains a finite ranked graph of word-and-tail effects. Checking each row certifies every expansion, installation and bootstrap. Its complete resource theorem has the form

$$(7.1) \quad \begin{aligned} m &\leq T_{\text{start}} + (u + 1)K_C(B_C + N_C + u)^2, \\ \text{space} &\leq F_{\text{start}} + Q_{\text{start}} + w_{\text{start}} + (u + 1)J_C(B_C + N_C + u) + 1. \end{aligned}$$

Here C is a fixed accepted bank/input certificate, and u is the number of steps of the source machine presented *to that bank*. If a frontend transformed the original source, u need not equal its original step count t . Startup, all refreshes, the partial final epoch and terminal work are included.

7.3. Difference and gap frontends. Deferred differences maintain a pending boundary correction and an orientation. If an original execution has t steps and r moves against the stored orientation, the native deferred execution has exactly $k = t + r$ steps, with $t \leq k \leq 2t$. The initial transformation adds one closing cell. Signed-gap compression then represents long blank stretches by finite signed-digit data; a checked quotient may merge states and symbols while preserving the reachable semantics.

The composed native frontend theorem gives

$$(7.2) \quad u \leq \text{boot} + 64(W_{\text{gap}} + 1)k, \quad t \leq k \leq 2t,$$

where W_{gap} is the proved compressed-run width of the deferred source. It depends on the history of marked records and logarithmic gap lengths. It is not uniformly logarithmic in original source time for arbitrary programs. Equation (7.2) counts native frontend transitions; a literal U20 bound must additionally compose it with the selected backend and its certificate-dependent constants.

The physical reader for this full path inverts the quotient, expands the signed gaps, applies the pending correction and integrates the differences. It uses seventeen tape roles and has explicit stage and handoff costs, given in Section 8. Writing an expanded source output costs time in the expanded output length. Compression does not make that output operation logarithmic.

7.4. Total checking and size selection. Let $C(T, x, a)$ be the total checked compiler result for candidate value a , including fallback when a check fails. Let $R(T, x)$ be the original-source reference result. The final policy returns the complete program/reader pair having the smaller represented initial size, keeping C on equality. Hence

$$(7.3) \quad |\mathcal{E}_T^{\min}(x, a)| = \min\{|C(T, x, a)|, |R(T, x)|\} \leq I_T(|x| + 1).$$

Both branches have halting equivalence and exact original-source physical readback. Correctness is independent of how the candidate value was produced.

This assertion does not say that a nonterminating candidate producer returns a value, or that every historical optimizer output is accepted. It

does not attach the reference execution bound to a smaller optimized tape. A uniform original-source runtime theorem for every optimized selection remains a distinct unproved claim.

7.5. Numerical scale. All four historical optimized inputs pass the existing complete validators. The binary writer example binds through a three-state deferred machine, a 218-state signed-gap machine, and a 141-state, eight-symbol quotient to a complete 1,239,701-cell initial U20 tape. That tape includes a 203,725-cell loader and expands an active bank of 6,816,447 cells. These are physical cells, not compressed bytes. Acceptance by the additional size guard has not been evaluated for these four historical fixtures.

The saved research accountant gives the following comparison at the same checkpoint of 100,000 original steps. It includes startup and retained history.

TABLE 5. Research cost calculations for the right-moving writer. These enormous complete runs were not executed literally and their exact numerical predictions are not universal Lean theorems.

Compilation	Target transitions	Retained physical cells
Balanced direct	1.06154×10^{27}	5.33908×10^{13}
Restricted difference/gap	1.50809×10^{30}	4.22953×10^{15}

The direct version is about 1,421 times faster at this checkpoint, according to that accountant. A compressed representation can lose at a finite horizon because of its larger compiled tables and startup. No crossover or all-input speed dominance has been established. These figures concern a nonhalting writer’s finite progress checkpoint, not a completed source halt. The exact 2,999-transition fixture has the stronger, separate status of a completed literal run checked in Lean.

7.6. Explicit startup budget. For a checked startup object S with helper depth d_S , initial roots R_S and loader bank P_S , its defined weight is

$$w_S = \text{calls}(d_S, R_S) \text{ budget}(P_S) + 2 + \text{budget}(P),$$

and its bound is

$$T_S = 30w_S(1 + \text{cells}(R_S, 0) + w_S).$$

The complete optimized resource theorem is **RestrictedRefresh.Startup.complete_resources**. For a fixed accepted bank, seed and stop certificate it bounds backend-source simulation time u by

$$T_S + (u + 1)K_C(B_C + N_C + u)^2.$$

It retains the whole loader in the physical space interval. The seed and all finite row certificates are premises supplied by successful checking, not informal assumptions about the Python producer.

8. PHYSICAL DECODER COSTS

8.1. Direct path. The direct path has five tape roles. With saved prefix length ℓ_{saved} , saved address count r_{saved} and source side lengths ℓ, r ,

$$D_0 = \ell_{\text{saved}} + r_{\text{saved}} + 4, \quad \text{workspace} \leq r_{\text{saved}} + \ell + r + 6.$$

All bounds include stage transitions and frontier blanks. The original U20 tape is read-only. The fixed table and that input tape are excluded from workspace, while address and output tapes remain allocated. The reader does not scan retired banks beyond its stopping boundary.

8.2. Full frontend path. The quotient adapter starts from the three physical source-output tapes of the direct reader. Let A, B be its encoded side widths and W the full represented quotient right length, including padding it scans. Its exact transition count is

$$2A + B + W + 7.$$

Let L, R be the expanded signed-gap side lengths, and e, f the counts of negative digits on the two encoded sides. Define

$$(8.1) \quad C_L = 2A + 5 + 32(A + 4)(2L + e + 1),$$

$$(8.2) \quad C_R = 3B + 9 + 32(B + 4)(2R + f + 1),$$

$$(8.3) \quad C_{\text{gap}} = C_L + C_R + L + 2R + 8,$$

$$(8.4) \quad C_{\text{front}} = 2A + B + W + 8 + C_{\text{gap}}.$$

The full physical reader satisfies

$$D \leq D_0 + 1 + C_{\text{front}}, \quad \text{prefixWorkspace} \leq 16 + 5D.$$

The integration substage alone takes exactly $L + 2R + 6$ transitions and at most $3L + 5R + 19$ represented cells at every prefix. Its accumulator belongs to the fixed finite source alphabet. Both orientations, all pending corrections and arbitrary designated blanks are covered. The right side is rewound before forward integration, and all copies and reversals are charged.

There are seventeen tape roles: the original read-only U20 tape, a retained address tape, and fifteen frontend roles. All old work buffers are retained at handoffs. The bound $16 + 5D$ is conservative and makes no workspace-optimality claim. It follows from the actual multi-tape transition system, not from treating expansion as a mathematical constant-time list operation.

9. THEOREM-TO-CLAIM MAP

The exact exported names below are resolved by the accompanying `verification/PaperChecks.lean`. Name resolution alone does not establish correspondence with a prose claim. The expanded contracts in `verification/ProofInterfaces.lean` provide the additional statement checks described below. File names denote sources in the frozen `utm20_lean/U20/` directory. Auxiliary names use the common U20 namespace.

Claim	File and principal theorem
20 defined instructions	<code>Machine.lean: instruction_count, undefined_exactly</code>
Literal queue operation	<code>RawProgram.lean: RawProgram.execute, RawProgram.bounded</code>
Exact local cost	<code>Cost.lean: Cost.slotTime_affine, Cost.slot_execution</code>
Complete finite fixture	<code>ResearchFixture.lean: ResearchFixture.exact_execution; Examples.lean: Examples.macro_cost_total</code>
Ordinary source step	<code>GuardedStep.lean: Guarded.source_cycle, Guarded.source_halt</code>
Physical adjacent lookup	<code>DirectPhase.lean: DirectPhase.forward, DirectPhase.returning</code>
Unconditional finite compiler	<code>DirectCompiler.lean: DirectAdapter.unitCertified; DirectStop.lean: DirectStop.compile</code>
Autonomous epoch	<code>RefreshAutonomous.lean: RefreshAutonomous.epoch, RefreshAutonomous.epochs</code>
Archive retention	<code>RefreshAutonomous.lean: RefreshAutonomous.Archives.retains</code>
Complete reference resources	<code>DirectResources.lean: DirectStop.complete_resources_standard</code>
Optimized startup resources	<code>StartupResources.lean: RestrictedRefresh.Startup.complete_resources</code>
Original-source physical output	<code>CompleteCompilerPhysical.lean: CompleteCompiler.complete_physical_output</code>
Uniform initial size	<code>SizeBoundedCompiler.lean: SizeBoundedCompiler.uniform_initial_size, SizeBoundedCompiler.initial_size</code>
Final universal interface	<code>FinalTheorem.lean: Finalized.finite_input_universality_and_readback</code>
Reference resource/readback witness	<code>FinalTheorem.lean: Finalized.reference_polynomial_physical</code>

9.1. Audit of the proof interfaces. The interface file contains eight public theorems and two private helpers. They use the namespace `U20.PaperInterfaces` and sit outside the frozen library. All ten were checked against a cache-free build of the unchanged formal archive on 2 October 2026, with a transitive dependency audit. The following checks compare their statements and definitions with the paper’s claims; they complement the library’s kernel-checked proofs.

- (1) *Input and fixed interpreter.* `source_input_convention` spells out the source tape at every integer, including empty input and blank padding. `reference_start_and_reader` checks target state A , head zero, and

a reader determined by the source table and its blank. The separate table comparison checks all 25 defined/undefined entries against the manuscript. The integer-tape transition definitions inspect the current symbol, write only the scanned cell and move by exactly one; coordinates are semantic data, not registers available to the twenty instructions.

- (2) *Unconditional compilation.* `arbitrary_candidate_contract` exposes finite support, linear represented initial size and halting equivalence as existence of finite source and target runs, together with expanded physical readback through the reader selected with that candidate. Its candidate argument has no correctness or future-termination premise. Inspection of `CompleteCompiler` confirms that acceptance checks the frontend, initial input, seed and output binding before attaching the backend; failure selects the reference construction. Generic simulation lemmas do have validity and finite-certificate premises. For the reference, `DirectAdapter.unitCertified` constructs these certificates; `DirectStop.compile` supplies the terminal. Their existence is not an assumption of the public theorem.
- (3) *One complete reference execution.* `reference_execution_contract` removes the abbreviation `BoundedRun`: for every $k \leq m$, an actual target prefix has its head between the stated endpoints and blank tape outside the stated support. This includes initialization, installations, the final partial epoch and the halted endpoint. Its transparent `PhysicalReadback` predicate then exhibits a reader run from that same target, a reader halt, equality of source state and head, pointwise equality at all integer tape coordinates, and every-prefix reader workspace. No independently chosen terminal tape enters this statement. The original proof identifies its witnesses using both terminal-time and terminal-configuration uniqueness.
- (4) *Explicit polynomial interval cardinality.* `reference_polynomial_space_contract` combines the actual initial-size bound with the endpoint estimate. With $S_T = I_T + 1 + J_T(B_T + 1)$ it states, for the same run and physical output,

$$(hi - lo + 1).toNat \leq S_T(t + 1)(\max(1, n) + t + 1).$$

The coefficient is a function of T alone. Its private arithmetic helper uses $n + 1 \leq \max(1, n) + t + 1$ and $B_T + \max(1, n) + t \leq (B_T + 1)(\max(1, n) + t + 1)$.

- (5) *Physical handoff and cost.* `halted_input_is_physical` checks the reflected-input identity under the proved terminal invariant. `halted_reader_cost` connects that same invariant to the exact $\ell_{\text{saved}} + r_{\text{saved}} + 4$ reader count and every-prefix workspace, retaining an arbitrary archive on the input tape. It keeps the validity and saved-label hypotheses visible through the terminal invariant. The direct compiler's `run_physical`

theorem constructs that invariant; no answer is supplied to reader execution. Observation interprets finite marked tapes. A separate binary numeral for the absolute head is outside the claim.

- (6) *Constants and scope.* `reference_coefficients` checks the displayed final time, space and initial-size coefficients by definitional equality. The epoch and frontend subexpressions were compared with their source definitions. All reference coefficients depend on T , whereas optimized backend coefficients belong to the accepted transformed pipeline. Initial-size minimization does not transfer the reference runtime bound to that pipeline. The generic reader bound $16 + 5D$ is in reader transitions D , not original source transitions t .

All interface contracts pass. They specify the output representation, start and input conventions, and the distinct costs of reader execution, target execution and compilation. The source-only interval corollary packages the asymptotic estimate in one execution statement. The reference runtime guarantee applies to the reference route; arbitrary candidates retain the separately stated guarantees.

10. FROZEN EVIDENCE AND INDEPENDENT REPRODUCTION

The formal companion is the immutable archive

`U20_Lean_Finalized_2026-09-29.zip`.

Its SHA-256, split only for typography, is

`2eff990c1bb52832cc13fddd9b019fa47
f1e73cf59a7319f32ecf4161f8be270`.

The archive is included in the Version 1 compendium under the DOI on the title page. The full 64-character string is in the submission check record and the archive’s adjacent checksum file. The archive has 847 payload members plus its internal release manifest, with no compiled Lean cache. The audit binds 418 source files and 226 verification logs by SHA-256. The completed audit time is 2026-09-29 15:42:55 UTC.

The reported audit counts are:

Measure	Count
Library source files, including root	360
Explicit theorem declarations	2,525
Theorem declarations, including generated	9,361
All audited declarations	15,313
Semantic mutation controls rejected	191
Forbidden-proof controls rejected	3
Optimized inputs accepted by existing validators	4
Preserved U21 source hashes unchanged	82

Allowed foundational axioms actually used are `propext`, `Classical.choice`, and `Quot.sound`. The transitive audit rejects `sorryAx`, custom

axioms and native-evaluation proof shortcuts. The small literal fixture is proved by kernel reduction. Mutation tests are additional fault-detection evidence, not logical premises of the universal theorems.

10.1. Proof-only reproduction. Use Lean version 4.32.2. The toolchain identifier is

```
leanprover/lean4:v4.32.2.
```

After extracting the archive, enter its `utm20_lean/` directory and run

```
lake build
```

```
lake env lean -DwarningAsError=true Audit.lean
```

The development uses Std and no external Lean packages. Installing the toolchain may need a network connection; checking the installed companion does not require a network service.

10.2. Full companion audit. Preserving an untouched extraction as the release record, run in a fresh copy:

```
python3 verify.py --clean --negative-controls \
  --proof-mutations --references
```

Python 3.10 or later suffices for the companion’s standard-library checks. This recreates logs and status files, including new timestamps. It builds and audits the library, compares hardware and fixtures, checks optimized certificates and their physical tape bindings, replays source/decoder tests and checks preserved provenance. The recorded final audit also performed a second fresh build and dependency audit from an isolated project without a compiled cache.

10.3. Paper-specific verification. The paper verifier reads the audited source tree. It checks the recorded source hashes, the 25-entry printed table, slot assembly and the complete 2,999-transition fixture. It resolves the named Lean interfaces, checks printed audit totals and benchmark figures against their saved reports, and checks PDF text, page count and local link destinations. Its JSON record separates these fresh manuscript checks from the original formal companion audit. All eight public contracts, including the polynomial-space execution contract, and two private helpers pass a transitive dependency audit. The compact coefficient lemmas and the exact template-export identity pass the same allowed-axiom check: 20 explicit compact-reference theorems and 32 theorem declarations including generated ones. For this submission, a fresh extraction containing no compiled Lean cache passed `lake build` and the complete `Audit.lean` dependency audit. The historical 191 semantic mutation controls and three forbidden-proof controls were not rerun for these checks; their counts above remain frozen evidence from 29 September.

The reproducible PDF build uses Tectonic 0.17.0 with the same 11-point `amsart`, Latin Modern and microtype style as the U21 version 1.0 submission. The two manuscripts are standalone $\text{\LaTeX}$ files with embedded

bibliographies where needed. Both sources are checked with the editor’s built-in compiler and have matching PDF exports with source hashes and page-rendering checks. The submission compendium retains the current manuscript, reproduction code and evidence, together with the complete immutable formal companion and its dependencies. Superseded editions and duplicate manuscript PDFs are excluded. PDF compilation and finite Python replay complement the quantified Lean proofs.

10.4. Reproducing the reference experiments. Run the paper verifier with

```
python3 verify_paper.py
```

Source-only verification is the default. It checks all eight paper-facing Lean files, regenerates the small arithmetic fixtures, replays both million-step prefixes and validates the source hashes of the saved bank exports. It also regenerates the static reader dictionaries, performs the small literal differential tests and completes both compressed macro replays through saved-tape readback. Every Lean source-check log records its command, exit status, source hash and an explicit pass or fail line. Replay reports distinguish the SHA-256 of each saved gzip file, used for file integrity, from the SHA-256 of its canonical uncompressed JSON, used for comparisons across compression libraries. Canonical JSON sorts object keys and uses ASCII escapes and compact separators; it retains the saved block representation. Recompression controls vary compression level and gzip timestamp without changing the content hash. Malformed terminal fields are rejected with descriptive reader errors. The bank exports are retained to avoid repeating the expensive host compilation on every verification run. To regenerate them, build `U20.DirectResources` with native objects, compile `ReferenceBenchmark.lean` and `ReferenceMetadata.lean` with Lean 4.32.2, and run their `zero` and `writer` modes. The README and `verification/reproduce_reference.py` provide the exact commands; the exporter has a 600-second limit per case and records failure explicitly. Native binaries are local build products, not proof certificates. The evaluator, bank descriptions, metadata, small arithmetic fixtures and result hashes are retained beside the paper sources.

11. BIBLIOGRAPHIC AND NUMERICAL PROVENANCE

The U21 style and comparison use the local version 1.0 submission and its formal companion. The Neary–Woods 2009 article is cited using the publisher’s bibliographic record: *Fundamenta Informaticae* 91(1), 123–144, DOI 10.3233/FI-2009-0036. The author-hosted PDF has a different header DOI and pagination; the university repository explicitly notes the incorrect PDF page numbers. Primary sources were consulted for the small-machine comparison. The public U21 version 1.0 is available at doi:10.5281/zenodo.22957167. Its DOI resolution, public repository

record and downloadable-file metadata were checked without authentication on 2 October 2026. The DOI for the Version 1 U20 submission is doi:10.5281/zenodo.23095173, also shown on this accompanying supplement’s title page. The compendium’s bibliographic evidence record preserves the inspection status and provenance of the comparison and formalization references, as reviewed on 30 September 2026. It records source inspection, not independent verification of the other authors’ proof artifacts.

The large writer figures come from the frozen research checkpoint report, not the Lean transition counter for a completed run. At 100,000 original writer steps, the final restricted difference/gap report records exactly

$$1,508,088,881,653,935,658,222,856,415,764$$

calculated target transitions and 4,229,534,598,213,983 retained cells. Its literal component checks and source-level replay are described in the research artifact. Table 5 reports these calculated costs with limited precision. They are separate from the main manuscript’s completed queue example and compressed reference-compiler replays. The paper verifier records the specific files and hashes used for this comparison.

12. ILLUSTRATION AND EXAMPLE CHECKS

The first-call diagram is generated from the actual table and fixture: all 25 configurations and unchanged exterior cells are checked. The quoted Lean conjunct is compared with its source after removing formatting. Kernel reduction proves the displayed two-letter copier schedule, including its separate fifth completion cycle, in `ReadabilityChecks.two_letter_schedule`. Run the focused checks with

```
python3 verification/verify_readability.py
```

The full source verifier also compiles that schedule and audits its dependencies. The ratio of proved bound to replayed cost is recomputed from the exact saved coefficients and completed trace. These are exposition checks, not new universality or performance claims.

13. CLAIMS INTENTIONALLY LEFT OPEN

Remaining questions include optimizer termination and complexity; sharp size and original-source runtime bounds for all optimized branches; certification of all accountant predictions; instruction-count minimality; full literal reference and reader runs; and polynomial host compilation in the source-table size. Selecting a candidate for smaller initial size alone does not establish the reference runtime bound. These stronger claims are independent of the reference universality theorem.

INDEPENDENT RESEARCHER