Round-Trip Composition of the Alphabet Transformations

Formal Proof in Isabelle/HOL

András Z. Salamon Michael Wehar

August 26, 2026

Abstract

Machine-checked formal proof in Isabelle/HOL that the two multitape tape-alphabet transformations compose *as machines*, in both orders, with the accepted language preserved and the composed running time bounded by an *explicit* affine function of the source machine's running time.

The two transformations are alphabet *enlargement* (alphabet_enlarge) and alphabet *reduction* (alphabet_reduce). Enlargement is the Hartmanis–Stearns / Hopcroft–Ullman linear-speedup construction [4, 6], which packs c contiguous tape cells into one block to run a machine c times faster. Reduction is the elementary per-symbol binary re-encoding of an arbitrary finite tape alphabet onto the fixed four-symbol alphabet sym4 — the folklore alphabet homomorphism [2, p. 23], set within the wider machine-simulation landscape by van Emde Boas and Papadimitriou [8, 7]. They are developed independently in the sibling sessions `Multitape_Alphabet_Enlargement` and `Multitape_Alphabet_Reduction` over the shared nondeterministic multitape substrate `Multitape_TM_Substrate`. This session (`Multitape_Alphabet_Roundtrip`) composes them.

For a well-formed source M with $|\Gamma_M| \geq 4$ and each order, the session establishes:

- *Language preservation*, an equivalence modulo the composed input re-encoding (`alphabet_reduce_enlarge_language` and `alphabet_enlarge_reduce_language`);
- a *composed linear-time bound with explicit constants* (`alphabet_reduce_enlarge_time_explicit` and `alphabet_enlarge_reduce_time_explicit`). Writing k for the source tape count and $b = \lceil \log_2 |\Gamma_M| \rceil$ for its per-symbol block width, reducing then enlarging runs in time $8(6k+1)\,b\,T(|w|) + 2\,b\,|w| + 4$, and enlarging then reducing in $8(6k+1)\,b'\,T(|w|+c-1) + 2(6k+1)\,b'\,|w| + 4(6k+1)\,b'$, where c is the block size and b' the block width of the enlarged alphabet. Each bound is also given, as a same-named corollary, in the

existential shape $\exists A, B, C.\ \text{time} \leq A\,T(|w|) + B\,|w| + C$ — these composed constants are established here, not inherited from a prior result.

Nothing is re-proved: each headline chains the two sibling sessions' existing language-preservation and explicit-time theorems (`alphabet_enlarge_time_explicit` and `alphabet_reduce_time_explicit`). The composed constants expose an order asymmetry invisible under $O(\cdot)$: the enlargement's block-speedup factor c *cancels* when the alphabet is reduced first but *survives* when it is enlarged first.

Acknowledgement. This formalisation was developed with proof engineering by Anthropic's Claude Opus 4.5, 4.6, 4.7, and 4.8 as the work progressed, together with other Claude family models in supporting roles. Claude Code was used to structure the work and for access to Isabelle/HOL. We also thank Arthur Freitas Ramos who provided feedback and also contributed several tooling improvements.

Contents

1 Overview

The substrate `Multitape_TM_Substrate` is nondeterministic by default: a machine's transition is a relation, and accepts_in_time_mttm $M\,w\,t$ means *some* run on w reaches the accept state within t steps. A tape-alphabet transformation is a total function on machines; each one changes only the alphabet and copies the tape count. The two transformations run opposite ways:

- M_e = alphabet_enlarge$M$ groups $c = |'c|$ contiguous cells into one block, so its tape alphabet is the block alphabet gamma_block Γ_M (all functions $'c \to \Gamma_M$, of cardinality $|\Gamma_M|^{\,c}$) and it simulates M with a factor-c speedup. Its input is re-encoded by encode_input $bl_M\,w$, of length $\lceil |w|/c \rceil$.
- M_r = alphabet_reduce$M$ spells each source symbol as a fixed-width block of sym4 = $\{\mathrm{BLANK4}, \mathrm{LE4}, \mathrm{BIT0}, \mathrm{BIT1}\}$ cells and simulates M with a slowdown logarithmic in $|\Gamma_M|$. Its input is re-encoded by encode_input_ar $\Gamma_M\,bl_M\,w$, of length exactly $b \cdot |w|$, where $b = \max(1, \lceil \log_2 |\Gamma_M| \rceil)$ is the per-symbol encoding width.

Because a composed term such as alphabet_reduce(alphabet_enlarge$M$) names constants from *both* sibling sessions, it cannot be stated inside either without a cross-import cycle (the sessions are deliberately independent: `Multitape_Alphabet_Enlargement` and `Multitape_Alphabet_Reduction` never import each other). `Multitape_Alphabet_Roundtrip` is a new session whose parent is `Multitape_TM_Substrate` and whose `ROOT` lists both siblings, giving the composition an honest home above both.

This round trip (applying enlargement and reduction to the same machine, in sequence) does not appear in the classical literature, where the two constructions are presented separately. It is at once a minimal worked example of using both transformations together and, through the explicit composed constants, a concrete illustration of how the *order* of two linear-time transformations changes the constant that $O(\cdot)$ would hide.

Terminology. The names *alphabet enlargement* and *alphabet reduction* are ours, chosen to make the duality explicit: one grows the tape alphabet to buy speed, the other shrinks it to a fixed size. In the classical literature the two carry different names and are not usually paired. Enlargement is the alphabet-growing half of the Hartmanis–Stearns tape-compression / linear-speedup theorem [4], given a textbook proof by Hopcroft and Ullman [6, Theorems 12.3–12.4]. Reduction is the folklore normalisation that a multitape machine over Γ is simulated over a fixed small alphabet at a

per-machine constant-factor cost of $\lceil \log_2 |\Gamma| \rceil$ cells per symbol (a *fixed* factor, not a $\log T$ slowdown; that slowdown belongs to the separate tape-count reduction); it is the alphabet homomorphism onto a two-or-more-symbol alphabet stated by Balcázar, Díaz, and Gabarró [2, p. 23], and sits within the general machine-model-simulation landscape surveyed by van Emde Boas [8] and Papadimitriou [7]. Classic texts *assume* the reduction rather than cost it. Hopcroft and Ullman [5, §7.2] invoke it only to justify a universal machine over a fixed alphabet, with no time analysis. The explicit constants below make visible a factor the classical presentations leave implicit.

Related AFP developments. The round trip itself — applying both alphabet transformations to one machine — has no formalised precedent; it composes the two sibling entries. Among the neighbouring Isabelle/HOL developments, Balbach's `Cook_Levin` [1] is the closest in spirit, since it too composes verified machine transformations (and its `Two_Four_Symbols` encoding is the closest prior to the reduction half), but it neither enlarges an alphabet for speed nor tracks the composed constants that reordering the two transformations exposes. Xu, Zhang, Urban, and Joosten's `Universal_Turing_Machine` [9] works over a fixed binary alphabet, and Dalvit and Thiemann's `Multitape_To_Singletape_TM` [3] — the substrate's model ancestor — fixes the tape count in the machine's type, whereas our substrate makes it a value, so the round trip is stated once for machines of every tape count.

2 The round-trip theorems

Throughout, M is a source machine with well_formed_mttm M and $|\Gamma_M| \geq 4$; $k = k_{\text{tm}}\, M$ is its tape count; $c = |'c|$ is the enlargement block size; and $b = \max(1, \lceil \log_2 |\Gamma_M| \rceil)$. Both combinators preserve the tape count (`alphabet_reduce_preserves_tape_count` and `k_tm_alphabet_enlarge`), so a single k threads the whole composition.

2.1 Language preservation

Each direction chains the two sibling equivalences, discharging one seam locally: the reduced machine is well-formed for any valid four-symbol source (`alphabet_reduce_well_formed`), and the enlarged machine's four-symbol lower bound follows from the constant-cell embedding of the source alphabet into the block alphabet (`card_gamma_block_ge_card`).

Reduce, then enlarge.

$$\begin{aligned}&\text{encode\_input}\, bl_{M_r}\\ &\quad (\text{encode\_input\_ar}\,\Gamma_M\, bl_M\, w) \in L\big(\text{alphabet\_enlarge}(\text{alphabet\_reduce}M)\big)\\ &\qquad \iff w \in L(M)\end{aligned}$$

(`alphabet_reduce_enlarge_language`).

Enlarge, then reduce.

$$\begin{aligned}&\text{encode\_input\_ar}\,\Gamma_{M_e}\, bl_{M_e}\\ &\quad (\text{encode\_input}\, bl_M\, w) \in L\big(\text{alphabet\_reduce}(\text{alphabet\_enlarge}M)\big)\\ &\qquad \iff w \in L(M)\end{aligned}$$

(`alphabet_enlarge_reduce_language`).

2.2 Composed time bounds, with explicit constants

The composed slowdown is obtained by routing the *inner* machine's running-time bound in as the *outer* transformation's per-input time function. Making the base bounds explicit makes the composed constants explicit too. The base bounds are: enlargement in $2\lceil |w|/c\rceil + 8\lceil T(|w|)/c\rceil + 4$ (`alphabet_enlarge_time_explicit`, constants independent of the machine), and reduction in $(6k+1)\, b\, T(|w|)$ (`alphabet_reduce_time_explicit`, with linear and additive terms zero).

Reduce, then enlarge (`alphabet_reduce_enlarge_time_explicit`). If M accepts w within $T(|w|)$ steps, then alphabet_enlarge(alphabet_reduce$M$) accepts the composed encoding within

$$8\,(6k+1)\, b\, T(|w|) \;+\; 2\, b\, |w| \;+\; 4$$

steps. The enlargement's block factor c *cancels*: it divides both the intermediate length and the routed running time, but the ceiling divisions are discarded in the relaxation to the affine form, so c does not appear. Reducing first fixes the alphabet at four symbols; the subsequent block speedup then buys nothing on the dominant term.

Enlarge, then reduce (`alphabet_enlarge_reduce_time_explicit`, assuming $\text{mono}\, T$**).** Writing $D = 6k+1$ and $b' = \max(1, \lceil c \cdot \log_2 |\Gamma_M|\rceil)$ (the per-symbol width of the *block* alphabet gamma_block Γ_M, of cardinality $|\Gamma_M|^c$): if M accepts w within $T(|w|)$ steps, then alphabet_reduce(alphabet_enlarge$M$) accepts the composed encoding within

$$8\, D\, b'\, T(|w| + c - 1) \;+\; 2\, D\, b'\, |w| \;+\; 4\, D\, b'$$

steps. Here c *does not* cancel. Enlarging first inflates the alphabet to $|\Gamma_M|^c$, so the reduction must spend $b' \approx c \cdot \log_2 |\Gamma_M|$ cells per symbol, exactly undoing the factor-c speedup on the dominant term, and c survives both in the time argument $T(|w| + c - 1)$ (the intermediate length is the lossy $\lceil |w|/c \rceil$, so mono T and a rounding to the next block boundary are needed) and, through b', in every coefficient.

The asymmetry. The two orders' leading T-coefficients are $8(6k+1)\,b$ for reduce-then-enlarge and $8(6k+1)\,b'$ for enlarge-then-reduce. Reducing first leaves this coefficient *independent of the block size* c: the alphabet is already fixed at four symbols, so the enlargement's factor-c speedup is discarded in the relaxation to the affine form and c drops out. Enlarging first instead inflates the alphabet to $|\Gamma_M|^c$, so its block width $b' = \lceil c \log_2 |\Gamma_M| \rceil \approx c\,b$ carries a factor of c into every coefficient, and the leading term grows to $\approx c \cdot 8(6k+1)\,b$. The same two transformations, composed in opposite orders, thus differ by a factor of about c in the leading constant — under $O(\cdot)$ both are simply "linear time" and the distinction is lost; the explicit constants recover it. These are upper bounds; matching lower bounds are future work (they require a minimum-displacement argument the one-directional simulation lemmas do not provide).

theory *AlphabetRoundtrip*
imports
Multitape_Alphabet_Enlargement.AlphabetEnlargement_Reverse
Multitape_Alphabet_Reduction.AlphabetReduction_Reverse
begin

3 Round-trip composition of the alphabet transformations

The two alphabet transformations compose *as machines* in both orders on the shared substrate. This theory states, for each order, language preservation and a composed linear-time slowdown bound. Nothing is re-proved: each headline chains the language / time theorems of *alphabet_enlarge* and *alphabet_reduce* already established in the two sibling sessions.

Two directions:

- **reduce after enlarge**: *alphabet_reduce* (*alphabet_enlarge M*). The enlarged machine feeds *alphabet_reduce*; the four-symbol lower bound its hypotheses require follows from the constant-cell embedding of the source alphabet into the block alphabet (*card_gamma_block_ge_card*).
- **enlarge after reduce**: *alphabet_enlarge* (*alphabet_reduce M*). The reduced machine is well-formed for *any* valid source with a four-symbol alphabet (*alphabet_reduce_well_formed*), which discharges the *well_formed_mttm* hypothesis *alphabet_enlarge* requires of its input – the cut-tolerance payoff.

3.1 The enlarged alphabet has at least as many symbols as the source

The block alphabet *gamma_block* Γ contains the constant blocks λ_. *a* for every $a \in \Gamma$, and the embedding sending *a* to that constant cell is injective (the index type is inhabited). Hence the enlarged alphabet is at least as large as the source, so a source with $4 \leq card\ \Gamma$ enlarges to a machine whose alphabet still meets the reduction combinator's minimal-alphabet bound.

lemma *card_gamma_block_ge_card*:
fixes Γ :: *'a set*
assumes *finG*: *finite* Γ
shows *card* Γ ≤ *card* (*gamma_block* Γ :: ((*'c* :: *enum*) ⇒ *'a*) *set*)
proof −
have *inj*: *inj_on* (λ*a*. (λ_::*'c*. *a*)) Γ
by (*rule inj_onI*) (*metis fun_cong*)
have *img*: (λ*a*. (λ_::*'c*. *a*)) ‘ Γ ⊆ *gamma_block* Γ

```
    by (auto simp: gamma_block_def)
  have card Γ = card ((λa. (λ_::'c. a)) ` Γ)
    by (simp add: card_image[OF inj])
  also have ... ≤ card (gamma_block Γ :: ('c ⇒ 'a) set)
    by (rule card_mono[OF finite_gamma_block[OF finG] img])
  finally show ?thesis .
qed
```

3.2 Time-composition preliminaries

A small fact used when composing the running-time bounds: the block count $(n + c - 1)\ div\ c$ (rounding $n\ /\ c$ up) that the enlargement bound divides by never exceeds n for a nonempty index ($0 < c$). (Weak acceptance monotonicity, formerly local here, is now the substrate's *accepts_in_time_mttm_mono*.)

```
lemma ceil_div_le:
  fixes n c :: nat
  assumes cpos: 0 < c
  shows (n + c - 1) div c ≤ n
proof (cases n = 0)
  case True
  from cpos obtain m where cm: c = Suc m using gr0_implies_Suc by blast
  have c - 1 < c using cm by simp
  hence (c - 1) div c = 0 by (rule div_less)
  thus ?thesis using True by simp
next
  case False
  then have n1: 1 ≤ n by simp
  have cnz: c ≠ 0 using cpos by simp
  have eq: n + c - 1 = (n - 1) + c using n1 by simp
  have (n + c - 1) div c = (n - 1) div c + 1
    unfolding eq by (rule div_add_self2[OF cnz])
  also have ... ≤ (n - 1) + 1
    using div_le_dividend[of n - 1 c] by simp
  also have ... = n using n1 by simp
  finally show ?thesis .
qed
```

3.3 Enlarge after reduce

Reduce M to the four-symbol machine *alphabet_reduce M*, then enlarge that. Language preservation chains the two headline biconditionals: *alphabet_reduce_language* (from M to the reduced machine, under the per-symbol encoding *encode_input_ar*) and *alphabet_enlarge_language* (from the reduced machine to its enlargement, under the block encoding *encode_input*). Two seams are discharged locally: the reduced machine is well-formed for any valid four-symbol source (*alphabet_reduce_well_formed*,

the input hypothesis *alphabet_enlarge* requires), and the encoded intermediate word lies in the reduced machine's input alphabet $\{BIT0,\ BIT1\}$ (*set_encode_input_ar*).

```
theorem alphabet_reduce_enlarge_language:
  fixes M :: ('q, 'a) mttm
  assumes vM: valid_mttm M
      and s_neq_t: s_tm M ≠ t_tm M
      and s_neq_r: s_tm M ≠ r_tm M
      and le_neq_bl: le_tm M ≠ bl_tm M
      and card_ge: card (Γ_tm M) ≥ 4
  shows ∀ w. set w ⊆ Sigma_tm M ⟶
          (encode_input (bl_tm (alphabet_reduce M))
              (encode_input_ar (Γ_tm M) (bl_tm M) w)
            ∈ Lang_mttm (alphabet_enlarge (alphabet_reduce M)
                :: (((('q × 'a ar_stage) × (sym4, 'c :: enum) ae_stage),
                    'c ⇒ sym4) mttm))
          = (w ∈ Lang_mttm M)
proof (intro allI impI)
  fix w assume w: set w ⊆ Sigma_tm M
  let ?Mr = alphabet_reduce M :: ('q × 'a ar_stage, sym4) mttm
  let ?enc = encode_input_ar (Γ_tm M) (bl_tm M) w
  have wfr: well_formed_mttm ?Mr
    by (rule alphabet_reduce_well_formed[OF vM card_ge])
  have red: (?enc ∈ Lang_mttm ?Mr) = (w ∈ Lang_mttm M)
    using alphabet_reduce_language[OF vM s_neq_t s_neq_r le_neq_bl card_ge]
w
    by blast
  have guard: set ?enc ⊆ Sigma_tm ?Mr
    using set_encode_input_ar[of Γ_tm M bl_tm M w]
    by (simp add: alphabet_reduce_Sigma)
  have enl: (encode_input (bl_tm ?Mr) ?enc
              ∈ Lang_mttm (alphabet_enlarge ?Mr
                  :: ((('q × 'a ar_stage) × (sym4, 'c) ae_stage),
                      'c ⇒ sym4) mttm))
            = (?enc ∈ Lang_mttm ?Mr)
    using alphabet_enlarge_language[OF wfr] guard by blast
  show (encode_input (bl_tm ?Mr) ?enc
          ∈ Lang_mttm (alphabet_enlarge ?Mr
              :: ((('q × 'a ar_stage) × (sym4, 'c) ae_stage),
                  'c ⇒ sym4) mttm))
       = (w ∈ Lang_mttm M)
    using enl red by simp
qed
```

The composed slowdown, with explicit constants. Reducing then enlarging is linear-time, with the affine bound $8 \cdot (6 \cdot k\_tm\ M + 1) \cdot b \cdot T(|w|) + 2 \cdot b \cdot |w| + 4$, where $b = block\_width\ (\Gamma\_tm\ M)$ is the per-symbol binary width and $k\_tm\ M$ the tape count. It routes the explicit reduction bound $((6 \cdot k\_tm\ M + 1) \cdot b \cdot T$, *alphabet_reduce_time_explicit*) in as the

enlargement's per-input running-time function *Tr*; on the encoded intermediate word (length $b \cdot |w|$) the enlargement hypothesis holds *exactly* ($Tr\ (b \cdot |w|) = (6 \cdot k\_tm\ M + 1) \cdot b \cdot T(|w|)$), and the enlargement's ceiling divisions are relaxed by *ceil_div_le* and *accepts_in_time_mttm_mono*. The enlargement's block-speedup divisor $c = card\ (UNIV :: 'c\ set)$ *cancels*: it divides the intermediate length and running time, but *ceil_div_le* discards it in the relaxation, so it does not appear in the bound — the block speedup buys nothing once the reduction has fixed the alphabet. The classical existential form is *alphabet_reduce_enlarge_time* below.

```
theorem alphabet_reduce_enlarge_time_explicit:
  fixes M :: ('q, 'a) mttm and T :: nat ⇒ nat
  assumes vM: valid_mttm M
      and s_neq_t: s_tm M ≠ t_tm M
      and s_neq_r: s_tm M ≠ r_tm M
      and le_neq_bl: le_tm M ≠ bl_tm M
      and card_ge: card (Γ_tm M) ≥ 4
  shows ∀ w. set w ⊆ Sigma_tm M ⟶
          accepts_in_time_mttm M w (T (length w)) ⟶
          accepts_in_time_mttm
            (alphabet_enlarge (alphabet_reduce M)
               :: (((('q × 'a ar_stage) × (sym4, 'c :: enum) ae_stage),
                    'c ⇒ sym4) mttm)
            (encode_input (bl_tm (alphabet_reduce M))
               (encode_input_ar (Γ_tm M) (bl_tm M) w))
            (8 * (6 * k_tm M + 1) * block_width (Γ_tm M) * T (length w)
              + 2 * block_width (Γ_tm M) * length w + 4)
proof −
  let ?Mr = alphabet_reduce M :: ('q × 'a ar_stage, sym4) mttm
  let ?Me = alphabet_enlarge ?Mr
              :: ((('q × 'a ar_stage) × (sym4, 'c :: enum) ae_stage),
                  'c ⇒ sym4) mttm
  let ?kf = block_width (Γ_tm M)
  let ?c = card (UNIV :: 'c set)
  let ?d = 6 * k_tm M + 1
  have wfr: well_formed_mttm ?Mr by (rule alphabet_reduce_well_formed[OF
vM card_ge])
  have kfpos: 0 < ?kf using block_width_pos[of Γ_tm M] by simp
  have cpos: 0 < ?c by (simp add: card_gt_0_iff)
  have ar: ∀ w. set w ⊆ Sigma_tm M ⟶
             accepts_in_time_mttm M w (T (length w)) ⟶
             accepts_in_time_mttm ?Mr (encode_input_ar (Γ_tm M) (bl_tm M)
w)
               (?d * ?kf * T (length w))
    by (rule alphabet_reduce_time_explicit[OF vM s_neq_t s_neq_r le_neq_bl
card_ge])
  — route the reduction bound in as the enlargement's per-input time function
  define Tr :: nat ⇒ nat where Tr = (λn. ?d * ?kf * T (n div ?kf))
  have ae: ∀ v. set v ⊆ Sigma_tm ?Mr ⟶
```

```
      accepts_in_time_mttm ?Mr v (Tr (length v)) ⟶
      accepts_in_time_mttm ?Me (encode_input (bl_tm ?Mr) v)
       (2 * ((length v + ?c - 1) div ?c)
        + 8 * ((Tr (length v) + ?c - 1) div ?c) + 4)
  by (rule alphabet_enlarge_time_explicit[OF wfr, where T = Tr])
 show ∀ w. set w ⊆ Sigma_tm M ⟶
      accepts_in_time_mttm M w (T (length w)) ⟶
      accepts_in_time_mttm ?Me
       (encode_input (bl_tm ?Mr) (encode_input_ar (Γ_tm M) (bl_tm M) w))
       (8 * (6 * k_tm M + 1) * block_width (Γ_tm M) * T (length w)
        + 2 * block_width (Γ_tm M) * length w + 4)
 proof (intro allI impI)
  fix w assume w: set w ⊆ Sigma_tm M
   and accM: accepts_in_time_mttm M w (T (length w))
  let ?v = encode_input_ar (Γ_tm M) (bl_tm M) w
  have lenv: length ?v = ?kf * length w
   by (simp add: length_encode_input_ar)
  have div_w: length ?v div ?kf = length w
   using lenv kfpos by simp
  have Tr_v: Tr (length ?v) = ?d * ?kf * T (length w)
   using div_w by (simp add: Tr_def)
  have accr: accepts_in_time_mttm ?Mr ?v (Tr (length ?v))
   using ar w accM Tr_v by simp
  have guard: set ?v ⊆ Sigma_tm ?Mr
   using set_encode_input_ar[of Γ_tm M bl_tm M w]
   by (simp add: alphabet_reduce_Sigma)
  have ae_acc: accepts_in_time_mttm ?Me (encode_input (bl_tm ?Mr) ?v)
                (2 * ((length ?v + ?c - 1) div ?c)
                 + 8 * ((Tr (length ?v) + ?c - 1) div ?c) + 4)
   using ae guard accr by blast
  have bound_le:
   2 * ((length ?v + ?c - 1) div ?c)
     + 8 * ((Tr (length ?v) + ?c - 1) div ?c) + 4
    ≤ 8 * ?d * ?kf * T (length w) + 2 * ?kf * length w + 4
  proof -
   have h1: (length ?v + ?c - 1) div ?c ≤ length ?v by (rule ceil_div_le[OF
cpos])
   have h2: (Tr (length ?v) + ?c - 1) div ?c ≤ Tr (length ?v) by (rule
ceil_div_le[OF cpos])
   have 2 * ((length ?v + ?c - 1) div ?c)
         + 8 * ((Tr (length ?v) + ?c - 1) div ?c) + 4
        ≤ 2 * length ?v + 8 * Tr (length ?v) + 4
    using h1 h2 by (simp add: add_mono mult_le_mono2)
   also have ... = 2 * (?kf * length w) + 8 * (?d * ?kf * T (length w)) + 4
    using lenv Tr_v by simp
   also have ... = 8 * ?d * ?kf * T (length w) + 2 * ?kf * length w + 4
    by (simp add: algebra_simps)
   finally show ?thesis .
  qed
```

```
    show accepts_in_time_mttm ?Me
          (encode_input (bl_tm ?Mr) ?v)
          (8 * (6 * k_tm M + 1) * block_width (Γ_tm M) * T (length w)
            + 2 * block_width (Γ_tm M) * length w + 4)
      by (rule accepts_in_time_mttm_mono[OF ae_acc bound_le])
  qed
qed
```

The classical existential form $A \cdot T(|w|) + B \cdot |w| + C$, with $A = 8 \cdot (6 \cdot k\_tm\ M + 1) \cdot b$, $B = 2 \cdot b$, and $C = 4$ from *alphabet_reduce_enlarge_time_explicit*.

```
theorem alphabet_reduce_enlarge_time:
  fixes M :: ('q, 'a) mttm and T :: nat ⇒ nat
  assumes vM: valid_mttm M
      and s_neq_t: s_tm M ≠ t_tm M
      and s_neq_r: s_tm M ≠ r_tm M
      and le_neq_bl: le_tm M ≠ bl_tm M
      and card_ge: card (Γ_tm M) ≥ 4
  obtains A B C :: nat
  where ∀ w. set w ⊆ Sigma_tm M ⟶
          accepts_in_time_mttm M w (T (length w)) ⟶
          accepts_in_time_mttm
            (alphabet_enlarge (alphabet_reduce M)
              :: (((('q × 'a ar_stage) × (sym4, 'c :: enum) ae_stage),
                   'c ⇒ sym4) mttm)
            (encode_input (bl_tm (alphabet_reduce M))
              (encode_input_ar (Γ_tm M) (bl_tm M) w))
            (A * T (length w) + B * length w + C)
proof (rule that[of 8 * (6 * k_tm M + 1) * block_width (Γ_tm M)
                    2 * block_width (Γ_tm M) 4])
  show ∀ w. set w ⊆ Sigma_tm M ⟶
          accepts_in_time_mttm M w (T (length w)) ⟶
          accepts_in_time_mttm
            (alphabet_enlarge (alphabet_reduce M)
              :: (((('q × 'a ar_stage) × (sym4, 'c :: enum) ae_stage),
                   'c ⇒ sym4) mttm)
            (encode_input (bl_tm (alphabet_reduce M))
              (encode_input_ar (Γ_tm M) (bl_tm M) w))
            (8 * (6 * k_tm M + 1) * block_width (Γ_tm M) * T (length w)
              + 2 * block_width (Γ_tm M) * length w + 4)
    by (rule alphabet_reduce_enlarge_time_explicit[OF vM s_neq_t s_neq_r
le_neq_bl card_ge])
qed
```

3.4 Reduce after enlarge

Enlarge M to the block machine *alphabet_enlarge M*, then reduce that back to four symbols. This is the direction whose seams need the enlarged machine's own well-formedness data: its start / accept / reject states, blank

and endmarker are the source ones tagged / block-encoded (the field accessors *s_tm_alphabet_enlarge* etc.), and its tape alphabet is the block alphabet *gamma_block* (Γ_*tm M*), whose cardinality is at least the source's (*card_gamma_block_ge_card*) — so a four-symbol source stays above the reduction's minimal-alphabet bound.

lemma *Gamma_tm_alphabet_enlarge*:
  fixes *M* :: (*'q*, *'a*) *mttm*
  shows Γ_*tm* (*alphabet_enlarge M*
                :: (*'q* × (*'a*, (*'c* :: *enum*)) *ae_stage*, *'c* ⇒ *'a*) *mttm*)
        = *gamma_block* (Γ_*tm M*)
  by (*cases M*) (*simp add*: *alphabet_enlarge_def*)

Enlargement preserves the tape count (the last *mttm* field is copied unchanged): the block transformation is purely alphabet-level, so the reduction's tape-count factor *6 · k_tm M + 1* is the *source* tape count even when the reduction is applied to the enlarged machine. Mirrors *alphabet_reduce_preserves_tape_count*.

lemma *k_tm_alphabet_enlarge*:
  fixes *M* :: (*'q*, *'a*) *mttm*
  shows *k_tm* (*alphabet_enlarge M*
                :: (*'q* × (*'a*, (*'c* :: *enum*)) *ae_stage*, *'c* ⇒ *'a*) *mttm*)
        = *k_tm M*
  by (*cases M*) (*simp add*: *alphabet_enlarge_def*)

Discharge of the reduction's input hypotheses on the enlarged machine (validity, the three non-degeneracy conditions, and the four-symbol lower bound), packaged for reuse by both the language and the time theorem.

lemma *alphabet_enlarge_reduce_hyps*:
  fixes *M* :: (*'q*, *'a*) *mttm*
  assumes *vM*: *valid_mttm M*
    and *s_neq_t*: *s_tm M* ≠ *t_tm M*
    and *s_neq_r*: *s_tm M* ≠ *r_tm M*
    and *le_neq_bl*: *le_tm M* ≠ *bl_tm M*
    and *card_ge*: *card* (Γ_*tm M*) ≥ *4*
  shows *valid_mttm* (*alphabet_enlarge M*
          :: (*'q* × (*'a*, *'c* :: *enum*) *ae_stage*, *'c* ⇒ *'a*) *mttm*)
   and *s_tm* (*alphabet_enlarge M*
          :: (*'q* × (*'a*, *'c*) *ae_stage*, *'c* ⇒ *'a*) *mttm*)
        ≠ *t_tm* (*alphabet_enlarge M*)
   and *s_tm* (*alphabet_enlarge M*
          :: (*'q* × (*'a*, *'c*) *ae_stage*, *'c* ⇒ *'a*) *mttm*)
        ≠ *r_tm* (*alphabet_enlarge M*)
   and *le_tm* (*alphabet_enlarge M*
          :: (*'q* × (*'a*, *'c*) *ae_stage*, *'c* ⇒ *'a*) *mttm*)
        ≠ *bl_tm* (*alphabet_enlarge M*)
   and *card* (Γ_*tm* (*alphabet_enlarge M*
          :: (*'q* × (*'a*, *'c*) *ae_stage*, *'c* ⇒ *'a*) *mttm*)) ≥ *4*
proof −

```
  let ?M' = alphabet_enlarge M :: ('q × ('a, 'c) ae_stage, 'c ⇒ 'a) mttm
  have finG: finite (Γ_tm M) by (rule valid_mttm_finite_Gamma[OF vM])
  show valid_mttm ?M' by (rule alphabet_enlarge_wf[OF vM])
  show s_tm ?M' ≠ t_tm ?M'
   using s_neq_t by (simp add: s_tm_alphabet_enlarge t_tm_alphabet_enlarge)
  show s_tm ?M' ≠ r_tm ?M'
   using s_neq_r by (simp add: s_tm_alphabet_enlarge r_tm_alphabet_enlarge)
  show le_tm ?M' ≠ bl_tm ?M'
  proof
    assume le_tm ?M' = bl_tm ?M'
    hence (LE_block (le_tm M) :: 'c ⇒ 'a) = bl_block (bl_tm M)
      by (simp add: le_tm_alphabet_enlarge bl_tm_alphabet_enlarge)
    hence le_tm M = bl_tm M by (rule LE_block_eq_bl_block_imp_eq)
    with le_neq_bl show False by simp
  qed
  have card (Γ_tm M) ≤ card (gamma_block (Γ_tm M) :: ('c ⇒ 'a) set)
    by (rule card_gamma_block_ge_card[OF finG])
  hence 4 ≤ card (gamma_block (Γ_tm M) :: ('c ⇒ 'a) set)
    using card_ge by linarith
  thus card (Γ_tm ?M') ≥ 4 by (simp add: Gamma_tm_alphabet_enlarge)
qed
```

The intermediate-word guard: the block encoding of a genuine input word lands in the enlarged machine's input alphabet (it is a block over the source, and avoids the two reserved blocks). Shared by the language and time theorems of this direction.

```
lemma encode_input_in_Sigma_alphabet_enlarge:
  fixes M :: ('q, 'a) mttm
  assumes vM: valid_mttm M and w: set w ⊆ Sigma_tm M
  shows set (encode_input (bl_tm M) w :: ('c :: enum ⇒ 'a) list)
          ⊆ Sigma_tm (alphabet_enlarge M
              :: ('q × ('a, 'c) ae_stage, 'c ⇒ 'a) mttm)
proof −
  have set (encode_input (bl_tm M) w :: ('c ⇒ 'a) list)
         ⊆ gamma_block (Sigma_tm M ∪ {bl_tm M})
    by (rule encode_input_in_gamma_block[OF w])
  moreover have bl_block (bl_tm M)
         ∉ set (encode_input (bl_tm M) w :: ('c ⇒ 'a) list)
    by (rule encode_input_no_bl_block[OF vM w])
  moreover have LE_block (le_tm M)
         ∉ set (encode_input (bl_tm M) w :: ('c ⇒ 'a) list)
    by (rule encode_input_no_LE_block[OF vM w])
  ultimately show ?thesis by (auto simp: Sigma_tm_alphabet_enlarge)
qed

theorem alphabet_enlarge_reduce_language:
  fixes M :: ('q, 'a) mttm
  assumes wfM: well_formed_mttm M
     and card_ge: card (Γ_tm M) ≥ 4
```

```
 shows ∀w. set w ⊆ Sigma_tm M ⟶
        (encode_input_ar
           (Γ_tm (alphabet_enlarge M
                  :: ('q × ('a, 'c :: enum) ae_stage, 'c ⇒ 'a) mttm))
           (bl_tm (alphabet_enlarge M))
           (encode_input (bl_tm M) w)
         ∈ Lang_mttm (alphabet_reduce (alphabet_enlarge M)
             :: ((('q × ('a, 'c) ae_stage) × ('c ⇒ 'a) ar_stage), sym4) mttm))
        = (w ∈ Lang_mttm M)
proof (intro allI impI)
 fix w assume w: set w ⊆ Sigma_tm M
 let ?M' = alphabet_enlarge M :: ('q × ('a, 'c) ae_stage, 'c ⇒ 'a) mttm
 have vM: valid_mttm M
   and s_neq_t: s_tm M ≠ t_tm M
   and s_neq_r: s_tm M ≠ r_tm M
   and le_neq_bl: le_tm M ≠ bl_tm M
   using wfM by auto
 note hyps = alphabet_enlarge_reduce_hyps[OF vM s_neq_t s_neq_r le_neq_bl
card_ge]
 have enl: (encode_input (bl_tm M) w ∈ Lang_mttm ?M') = (w ∈ Lang_mttm
M)
   using alphabet_enlarge_language[OF wfM] w by blast
 have guard: set (encode_input (bl_tm M) w :: ('c ⇒ 'a) list) ⊆ Sigma_tm ?M'
   by (rule encode_input_in_Sigma_alphabet_enlarge[OF vM w])
 have red: (encode_input_ar (Γ_tm ?M') (bl_tm ?M') (encode_input (bl_tm
M) w)
             ∈ Lang_mttm (alphabet_reduce ?M'
                 :: ((('q × ('a, 'c) ae_stage) × ('c ⇒ 'a) ar_stage), sym4) mttm))
          = (encode_input (bl_tm M) w ∈ Lang_mttm ?M')
  using alphabet_reduce_language[OF hyps(1) hyps(2) hyps(3) hyps(4) hyps(5)]
guard
   by blast
 show (encode_input_ar (Γ_tm ?M') (bl_tm ?M') (encode_input (bl_tm M) w)
         ∈ Lang_mttm (alphabet_reduce ?M'
             :: ((('q × ('a, 'c) ae_stage) × ('c ⇒ 'a) ar_stage), sym4) mttm))
       = (w ∈ Lang_mttm M)
   using red enl by simp
qed
```

The composed slowdown for reduce-after-enlarge, with explicit constants. Unlike the other order, the inner transformation (enlargement) *speeds up* by the block factor $c = card\ (UNIV :: 'c\ set)$, so the intermediate word has length $\lceil |w| \ / \ c \rceil$ — a lossy function of $|w|$, not an exact multiple — and the enlarged machine's running time cannot be routed into the reduction's time function exactly. Two mild ingredients close the gap: T is assumed non-decreasing ($mono\ T$), and the outer bound is stated at $T(|w| + c - 1)$ — T at the input length rounded up to the next block boundary. The bound is $8 \cdot D \cdot b' \cdot T(|w| + c - 1) + 2 \cdot D \cdot b' \cdot |w| + 4 \cdot D \cdot b'$, where $D = 6 \cdot k\_tm\ M + 1$ (the tape count is preserved by enlargement,

k_tm_alphabet_enlarge) and $b' =$ *block_width* (Γ_*tm* (*alphabet_enlarge M*)) = *block_width* (*gamma_block* (Γ_*tm M*)) is the per-symbol width of the *block* alphabet, i.e. $\lceil c \cdot log_2\ (card\ \Gamma_M) \rceil$. Because b' grows with c, the block-speedup factor *does not* cancel here: c survives both in the time argument $T(|w| + c - 1)$ and, through b', in every coefficient. The classical existential form is *alphabet_enlarge_reduce_time* below.

theorem *alphabet_enlarge_reduce_time_explicit*:
  fixes *M* :: ('*q*, '*a*) *mttm* **and** *T* :: *nat* ⇒ *nat*
  assumes *wfM*: *well_formed_mttm M*
    and *card_ge*: *card* (Γ_*tm M*) ≥ *4*
    and *Tmono*: *mono T*
  shows ∀ *w*. *set w* ⊆ *Sigma_tm M* ⟶
      accepts_in_time_mttm M w (*T* (*length w*)) ⟶
      accepts_in_time_mttm
        (*alphabet_reduce* (*alphabet_enlarge M*)
          :: ((('*q* × ('*a*, '*c* :: *enum*) *ae_stage*) × ('*c* ⇒ '*a*) *ar_stage*), *sym4*) *mttm*)
        (*encode_input_ar*
            (Γ_*tm* (*alphabet_enlarge M* :: ('*q* × ('*a*, '*c*) *ae_stage*, '*c* ⇒ '*a*) *mttm*))
          (*bl_tm* (*alphabet_enlarge M*))
          (*encode_input* (*bl_tm M*) *w*))
        (*8* ∗ (*6* ∗ *k_tm M* + *1*)
            ∗ *block_width* (Γ_*tm* (*alphabet_enlarge M*
                :: ('*q* × ('*a*, '*c*) *ae_stage*, '*c* ⇒ '*a*) *mttm*))
            ∗ *T* (*length w* + *card* (*UNIV* :: '*c set*) − *1*)
          + *2* ∗ (*6* ∗ *k_tm M* + *1*)
            ∗ *block_width* (Γ_*tm* (*alphabet_enlarge M*
                :: ('*q* × ('*a*, '*c*) *ae_stage*, '*c* ⇒ '*a*) *mttm*))
            ∗ *length w*
          + *4* ∗ (*6* ∗ *k_tm M* + *1*)
            ∗ *block_width* (Γ_*tm* (*alphabet_enlarge M*
                :: ('*q* × ('*a*, '*c*) *ae_stage*, '*c* ⇒ '*a*) *mttm*)))
proof −
  let *?M'* = *alphabet_enlarge M* :: ('*q* × ('*a*, '*c*) *ae_stage*, '*c* ⇒ '*a*) *mttm*
  let *?c* = *card* (*UNIV* :: '*c set*)
  let *?ke* = *block_width* (Γ_*tm ?M'*)
  let *?d* = *6* ∗ *k_tm M* + *1*
  have *vM*: *valid_mttm M*
    and *s_neq_t*: *s_tm M* ≠ *t_tm M*
    and *s_neq_r*: *s_tm M* ≠ *r_tm M*
    and *le_neq_bl*: *le_tm M* ≠ *bl_tm M*
    using *wfM* **by** *auto*
  have *cpos*: *0* < *?c* **by** (*simp add*: *card_gt_0_iff*)
  have *ktm*: *k_tm ?M'* = *k_tm M* **by** (*rule k_tm_alphabet_enlarge*)
  note *hyps* = *alphabet_enlarge_reduce_hyps*[*OF vM s_neq_t s_neq_r le_neq_bl card_ge*]
  — enlargement time constants (explicit: *al* = *2*, *fe* = *4*)

```
  have ae: ∀ w. set w ⊆ Sigma_tm M ⟶
        accepts_in_time_mttm M w (T (length w)) ⟶
        accepts_in_time_mttm ?M' (encode_input (bl_tm M) w)
          (2 * ((length w + ?c − 1) div ?c)
            + 8 * ((T (length w) + ?c − 1) div ?c) + 4)
    by (rule alphabet_enlarge_time_explicit[OF wfM])
  — route the enlarged machine's running time in as the reduction's time function
  define Te :: nat ⇒ nat where Te = (λn. 8 * T (?c * n) + 2 * n + 4)
  — reduction on the enlarged machine (explicit: e = fr = 0, d = 6 · k_tm M +
1 by tape preservation, b the block-alphabet width)
  have ar0: ∀ v. set v ⊆ Sigma_tm ?M' ⟶
        accepts_in_time_mttm ?M' v (Te (length v)) ⟶
        accepts_in_time_mttm (alphabet_reduce ?M'
           :: (((′q × (′a, ′c) ae_stage) × (′c ⇒ ′a) ar_stage), sym4) mttm)
          (encode_input_ar (Γ_tm ?M') (bl_tm ?M') v)
          ((6 * k_tm ?M' + 1) * ?ke * Te (length v))
    by (rule alphabet_reduce_time_explicit[OF hyps(1) hyps(2) hyps(3) hyps(4)
hyps(5)])
  have ar: ∀ v. set v ⊆ Sigma_tm ?M' ⟶
        accepts_in_time_mttm ?M' v (Te (length v)) ⟶
        accepts_in_time_mttm (alphabet_reduce ?M'
           :: (((′q × (′a, ′c) ae_stage) × (′c ⇒ ′a) ar_stage), sym4) mttm)
          (encode_input_ar (Γ_tm ?M') (bl_tm ?M') v)
          (?d * ?ke * Te (length v))
    using ar0 by (simp add: ktm)
  show ∀ w. set w ⊆ Sigma_tm M ⟶
        accepts_in_time_mttm M w (T (length w)) ⟶
        accepts_in_time_mttm (alphabet_reduce ?M'
           :: (((′q × (′a, ′c) ae_stage) × (′c ⇒ ′a) ar_stage), sym4) mttm)
          (encode_input_ar (Γ_tm ?M') (bl_tm ?M') (encode_input (bl_tm M)
w))
          (8 * ?d * ?ke * T (length w + ?c − 1)
            + 2 * ?d * ?ke * length w + 4 * ?d * ?ke)
  proof (intro allI impI)
    fix w assume w: set w ⊆ Sigma_tm M
      and accM: accepts_in_time_mttm M w (T (length w))
    let ?v = encode_input (bl_tm M) w :: (′c ⇒ ′a) list
    have lenv: length ?v = (length w + ?c − 1) div ?c
      by (rule length_encode_input)
    have lv_le: length ?v ≤ length w
      using lenv ceil_div_le[OF cpos] by simp
    have cge: length w ≤ ?c * length ?v
    proof −
      have eq: ?c * ((length w + ?c − 1) div ?c) + (length w + ?c − 1) mod ?c
            = length w + ?c − 1
        by (rule mult_div_mod_eq)
      have ml: (length w + ?c − 1) mod ?c < ?c
        using cpos by (rule mod_less_divisor)
      have length w ≤ ?c * ((length w + ?c − 1) div ?c)
```

```
    using eq ml cpos by linarith
  thus ?thesis by (simp add: lenv)
qed
have mcle: ?c * length ?v ≤ length w + ?c - 1
proof -
  have ?c * ((length w + ?c - 1) div ?c) ≤ length w + ?c - 1
    by (metis mult_div_mod_eq le_add1)
  thus ?thesis using lenv by simp
qed
— enlargement: the enlarged machine accepts the block encoding
have accAE: accepts_in_time_mttm ?M' ?v
              (2 * ((length w + ?c - 1) div ?c)
                + 8 * ((T (length w) + ?c - 1) div ?c) + 4)
  using ae w accM by blast
— its bound is dominated by Te at the intermediate length
have le1: 2 * ((length w + ?c - 1) div ?c)
            + 8 * ((T (length w) + ?c - 1) div ?c) + 4
          ≤ Te (length ?v)
proof -
  have alv: 2 * ((length w + ?c - 1) div ?c) = 2 * length ?v
    by (simp add: lenv)
  have tb: 8 * ((T (length w) + ?c - 1) div ?c) ≤ 8 * T (?c * length ?v)
  proof -
    have (T (length w) + ?c - 1) div ?c ≤ T (length w)
      by (rule ceil_div_le[OF cpos])
    also have ... ≤ T (?c * length ?v) using Tmono cge by (rule monoD)
    finally show ?thesis by (rule mult_le_mono2)
  qed
  have 2 * ((length w + ?c - 1) div ?c)
         + 8 * ((T (length w) + ?c - 1) div ?c) + 4
       = 2 * length ?v + 8 * ((T (length w) + ?c - 1) div ?c) + 4
    by (simp only: alv)
  also have ... ≤ 2 * length ?v + 8 * T (?c * length ?v) + 4
    using tb by simp
  also have ... = Te (length ?v) by (simp add: Te_def)
  finally show ?thesis .
qed
have accr_hyp: accepts_in_time_mttm ?M' ?v (Te (length ?v))
  by (rule accepts_in_time_mttm_mono[OF accAE le1])
have guard: set ?v ⊆ Sigma_tm ?M'
  by (rule encode_input_in_Sigma_alphabet_enlarge[OF vM w])
have accAR: accepts_in_time_mttm (alphabet_reduce ?M'
              :: ((('q × ('a, 'c) ae_stage) × ('c ⇒ 'a) ar_stage), sym4) mttm)
              (encode_input_ar (Γ_tm ?M') (bl_tm ?M') ?v)
              (?d * ?ke * Te (length ?v))
  using ar guard accr_hyp by blast
have le2: ?d * ?ke * Te (length ?v)
          ≤ 8 * ?d * ?ke * T (length w + ?c - 1)
              + 2 * ?d * ?ke * length w + 4 * ?d * ?ke
```

```
    proof −
      have T_le: T (?c * length ?v) ≤ T (length w + ?c − 1)
        using Tmono mcle by (simp add: monoD)
      have TeB: Te (length ?v) ≤ 8 * T (length w + ?c − 1) + 2 * length w + 4
        unfolding Te_def using T_le lv_le
        by (auto intro: add_mono mult_le_mono2)
      have ?d * ?ke * Te (length ?v)
            ≤ ?d * ?ke * (8 * T (length w + ?c − 1) + 2 * length w + 4)
        using TeB by (rule mult_le_mono2)
      also have ... = 8 * ?d * ?ke * T (length w + ?c − 1)
                    + 2 * ?d * ?ke * length w + 4 * ?d * ?ke
        by (simp add: algebra_simps)
      finally show ?thesis .
    qed
    show accepts_in_time_mttm (alphabet_reduce ?M'
           :: (((′q × (′a, ′c) ae_stage) × (′c ⇒ ′a) ar_stage), sym4) mttm)
         (encode_input_ar (Γ_tm ?M') (bl_tm ?M') ?v)
         (8 * ?d * ?ke * T (length w + ?c − 1)
           + 2 * ?d * ?ke * length w + 4 * ?d * ?ke)
      by (rule accepts_in_time_mttm_mono[OF accAR le2])
  qed
qed
```

The classical existential form $A \cdot T(|w| + c - 1) + B \cdot |w| + C$, with $A = 8 \cdot D \cdot b'$, $B = 2 \cdot D \cdot b'$, $C = 4 \cdot D \cdot b'$ for $D = 6 \cdot k\_tm\ M + 1$ and $b' = block\_width\ (\Gamma\_tm\ (alphabet\_enlarge\ M))$, from *alphabet_enlarge_reduce_time_explicit*.

```
theorem alphabet_enlarge_reduce_time:
  fixes M :: (′q, ′a) mttm and T :: nat ⇒ nat
  assumes wfM: well_formed_mttm M
      and card_ge: card (Γ_tm M) ≥ 4
      and Tmono: mono T
  obtains A B C :: nat
  where ∀ w. set w ⊆ Sigma_tm M ⟶
          accepts_in_time_mttm M w (T (length w)) ⟶
          accepts_in_time_mttm
            (alphabet_reduce (alphabet_enlarge M)
               :: (((′q × (′a, ′c :: enum) ae_stage) × (′c ⇒ ′a) ar_stage), sym4)
mttm)
            (encode_input_ar
                 (Γ_tm (alphabet_enlarge M :: (′q × (′a, ′c) ae_stage, ′c ⇒ ′a)
mttm))
               (bl_tm (alphabet_enlarge M))
               (encode_input (bl_tm M) w))
            (A * T (length w + card (UNIV :: ′c set) − 1) + B * length w + C)
proof (rule that[of 8 * (6 * k_tm M + 1)
                    * block_width (Γ_tm (alphabet_enlarge M
                        :: (′q × (′a, ′c) ae_stage, ′c ⇒ ′a) mttm))
                  2 * (6 * k_tm M + 1)
```

```
                    * block_width (Γ_tm (alphabet_enlarge M
                        :: ('q × ('a, 'c) ae_stage, 'c ⇒ 'a) mttm))
                4 * (6 * k_tm M + 1)
                    * block_width (Γ_tm (alphabet_enlarge M
                        :: ('q × ('a, 'c) ae_stage, 'c ⇒ 'a) mttm))])
  show ∀ w. set w ⊆ Sigma_tm M ⟶
        accepts_in_time_mttm M w (T (length w)) ⟶
        accepts_in_time_mttm
          (alphabet_reduce (alphabet_enlarge M)
            :: ((('q × ('a, 'c) ae_stage) × ('c ⇒ 'a) ar_stage), sym4) mttm)
          (encode_input_ar
                (Γ_tm (alphabet_enlarge M :: ('q × ('a, 'c) ae_stage, 'c ⇒ 'a)
mttm))
              (bl_tm (alphabet_enlarge M))
              (encode_input (bl_tm M) w))
          (8 * (6 * k_tm M + 1)
              * block_width (Γ_tm (alphabet_enlarge M
                  :: ('q × ('a, 'c) ae_stage, 'c ⇒ 'a) mttm))
              * T (length w + card (UNIV :: 'c set) - 1)
            + 2 * (6 * k_tm M + 1)
              * block_width (Γ_tm (alphabet_enlarge M
                  :: ('q × ('a, 'c) ae_stage, 'c ⇒ 'a) mttm))
              * length w
            + 4 * (6 * k_tm M + 1)
              * block_width (Γ_tm (alphabet_enlarge M
                  :: ('q × ('a, 'c) ae_stage, 'c ⇒ 'a) mttm)))
    by (rule alphabet_enlarge_reduce_time_explicit[OF wfM card_ge Tmono])
qed

end
```

References

[1] F. J. Balbach. The Cook-Levin theorem. *Archive of Formal Proofs*, January 2023. https://isa-afp.org/entries/Cook_Levin.html, Formal proof development.

[2] J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1988.

[3] C. Dalvit and R. Thiemann. A verified translation of multitape Turing machines into singletape Turing machines. *Archive of Formal Proofs*, November 2022. https://isa-afp.org/entries/Multitape_To_Singletape_TM.html, Formal proof development.

[4] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the AMS*, 117:285–306, 1965.

[5] J. E. Hopcroft and J. D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley, 1969.

[6] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[7] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[8] P. van Emde Boas. Machine models and simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 1–66. Elsevier, 1990.

[9] J. Xu, X. Zhang, C. Urban, S. J. C. Joosten, and F. Regensburger. Universal Turing machine. *Archive of Formal Proofs*, February 2019. https://isa-afp.org/entries/Universal_Turing_Machine.html, Formal proof development.